

# **JAVA BÁSICO** **PARA APRENDICES**

**LIBRO JAVA BÁSICO PARA APRENDICES**

ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hf6e-nj87>



# JAVA BÁSICO PARA APRENDICES

**Primera Edición 2021 Vol. 1**

**Editorial EIDEC**

Sello Editorial EIDEC (978-958-53018)

NIT 900583173-1

**ISBN:** 978-958-53018-8-7

**Formato:** Digital PDF (Portable Document Format)

**DOI:** <https://doi.org/10.34893/hf6e-nj87>

**Publicación:** Colombia

**Fecha Publicación:** 2021-05-31

**Coordinación Editorial**

Escuela Internacional de Negocios y Desarrollo Empresarial de Colombia – EIDEC

Universidad Enrique Guzman y Valle – Peru

Centro de Investigación Científica, Empresarial y Tecnológica de Colombia – CEINCET

Red de Investigación en Educación, Empresa y Sociedad – REDIEES

**Revisión y pares evaluadores**

Centro de Investigación Científica, Empresarial y Tecnológica de Colombia – CEINCET

Red de Investigación en Educación, Empresa y Sociedad – REDIEES

**Entidad Financiadora**

Universidad Enrique Guzmán y Valle – Perú



*LIBRO JAVA BÁSICO PARA APRENDICES*

ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hf6e-nj87>



## **Coordinadores editoriales**

Dra. Kriss Melody Calla Vásquez

**Universidad Enrique Guzmán y Valle – Perú**

Mg. Yohanna Milena Rueda Mahecha

**Editorial EIDEC - Colombia**

Dr. Cesar Augusto Silva Giraldo

**Centro de Investigación Científica, Empresarial y Tecnológica de Colombia – CEINCET – Colombia.**

Dr. David Andrés Suarez Suarez

**Red de Investigación en Educación, Empresa y Sociedad – REDIEES – Colombia.**

El libro **JAVA BÁSICO PARA APRENDICES**, esta publicado bajo la licencia de Creative Commons Atribución-NoComercial 4.0 Internacional (CC BY-NC 4.0) Internacional (<https://creativecommons.org/licenses/by-nc/4.0/deed.es>). Esta licencia permite copiar, adaptar, redistribuir y reproducir el material en cualquier medio o formato, con fines no comerciales, dando crédito al autor y fuente original, proporcionando un enlace de la licencia de Creative Commons e indicando si se han realizado cambios.

**Licencia: CC BY-NC 4.0.**

**NOTA EDITORIAL:** Las opiniones y los contenidos de los resúmenes publicados en el libro **JAVA BÁSICO PARA APRENDICES**. Son de responsabilidad exclusiva de los autores; así mismo, éstos se responsabilizarán de obtener el permiso correspondiente para incluir material publicado por parte de la **Editorial EIDEC** y la entidad financiadora de la publicación **Universidad Enrique Guzmán y Valle – Perú**.



**LIBRO JAVA BÁSICO PARA APRENDICES**

ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hf6e-nj87>



# JAVA BÁSICO PARA APRENDICES

MSc. Dr. Manuel Jesús Abanto Morales

Dra. Kriss Melody Calla Vásquez

Dr. Ilich Iván Pumacayo Palomino

Dra. Julia Lizet Torres Rivera

Dra. Eliana Castañeda Núñez

Lic. Nathaly Leslie Calla Vásquez



**LIBRO JAVA BÁSICO PARA APRENDICES**

ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hf6e-nj87>



## **Dedicatoria**

Con todo cariño para mis padres que están en el reino del señor, para mis hermanos, mi esposa Yolanda, para mis hijas Carol, Kattia y para mis nietos.

MSc. Dr. Manuel Jesús Abanto Morales





**LIBRO JAVA BÁSICO PARA APRENDICES**

ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hf6e-nj87>



## Prólogo

Con la ilusión de aportar en el aprendizaje de la programación de computadoras con el lenguaje de programación de objetos Java, se desarrolló el presente libro que permitirá el autoaprendizaje desde cero utilizando el IDE NETBEANS, los temas que se desarrollan son:

Capítulo 1. Introducción a Java

Capítulo 2. Sintaxis en programas JAVA

Capítulo 3. Tipo de datos en JAVA

Capítulo 4. Estructuras de control y arrays en JAVA

Capítulo 5. Clases en Java

Capítulo 6. Herencia e interfaces en JAVA

Capítulo 7. Excepciones en JAVA

**El desarrollo de los capítulos incluye ejercicios de programas desarrollados en Netbeans en forma completa.**



**LIBRO JAVA BÁSICO PARA APRENDICES**

ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hf6e-nj87>



# Contenido

|   |    |
|---|----|
| 1. INTRODUCCIÓN A JAVA.....                         | 17 |
| 1.1. Historia .....                                 | 18 |
| 1.2. Definición de java .....                       | 18 |
| 1.3. Características del lenguaje Java .....        | 18 |
| 1.4. Herramientas de desarrollo.....                | 20 |
| 1.5. Entorno Java para desarrollo .....             | 20 |
| 1.6. El típico primer programa .....                | 21 |
| 2. SINTAXIS EN PROGRAMAS JAVA.....                  | 30 |
| 2.1. Comentarios.....                               | 31 |
| 2.2 Tokens .....                                    | 32 |
| 2.2.1. Identificadores .....                        | 32 |
| 2.2.2. Palabras reservadas.....                     | 33 |
| 2.2.3 Separadores Los paréntesis ():.....           | 33 |
| 2.2.4 Operadores Java.....                          | 34 |
| 2.2.4.1 Operadores de asignación Java operador..... | 34 |
| 2.2.4.2 Operadores aritméticos Java.....            | 36 |
| 2.2.4.3 Operadores relacionales Java .....          | 36 |
| 2.2.4.4 Operadores lógicos Java.....                | 36 |



|   |    |
|---|----|
| 3. TIPOS DE DATOS EN JAVA .....   | 38 |
| 3.1. Tipos de datos primitivos. ....  | 38 |
| 3.1.1. Conversión de tipos .....  | 40 |
| 3.2. Tipos de datos referencia.....   | 41 |
| 3.3. Recolector de basura. ....   | 43 |
| 3.4. Declaración de variables. ....   | 44 |
| 3.5. Ámbito de las variables. ....  | 44 |
| 3.6. Ejercicios .....   | 47 |
| 4. ESTRUCTURAS DE CONTROL EN JAVA .....   | 48 |
| 4.1. Estructuras condicionales.....   | 48 |
| 4.1.1. Bifurcación: if-else, if-else-if .....                                     | 48 |
| 4.1.2. Selección múltiple: switch. ....   | 50 |
| 4.2. Estructuras de repetición. ....  | 53 |
| 4.2.1. Repetición sobre un rango determinado. for.....                            | 53 |
| 4.2.2. Repeticiones condicionales: while, do while. Sentencia while: ...          | 55 |
| 4.2.3. Uso de break y continue.....   | 57 |
| 4.3. Recursividad. ....   | 59 |
| 4.4. Pasos para desarrollar e implementar un programa orientado a<br>objetos..... | 62 |
| 4.4.1. Análisis del problema. ....  | 63 |
| 4.4.2. Diseño del programa. ....  | 65 |



|   |    |
|---|----|
| 4.4.3. Programación. ....                         | 67 |
| 4.4.4. Implementación. ....                       | 67 |
| 4.5. Arrays. ....                                 | 68 |
| 4.6. Cadenas de caracteres. ....                  | 73 |
| 4.7 Ejercicios .....                              | 73 |
| 5. CLASES EN JAVA .....                           | 75 |
| 5.1. Definición de una clase en Java.....         | 75 |
| 5.2. Atributos. ....                              | 76 |
| 5.3. Métodos estáticos o de clase. ....           | 78 |
| 5.4. Constructores .....                          | 81 |
| 5.5 Creación de objetos .....                     | 84 |
| 5.5. Paso por valor y paso por referencia.....    | 86 |
| 5.6. Sobrecarga de Métodos .....                  | 88 |
| 5.7. Finalización .....                           | 88 |
| 5.8 Ejercicios .....                              | 89 |
| 6. HERENCIA E INTERFACE EN JAVA .....             | 91 |
| 6.1. Herencia.....                                | 91 |
| 6.1.1. Sobrescritura de variables y métodos. .... | 96 |
| 6.1.2. Sobrescritura de constructores.....        | 98 |
| 6.1.3. Vinculación dinámica. ....                 | 98 |
| 6.1.4. El operador instanceof. ....               | 99 |



|   |     |
|---|-----|
| 6.1.1. Clases abstractas.....                         | 102 |
| 6.2. Interfaces.....                                  | 103 |
| 6.3. Paquetes.....                                    | 109 |
| 6.4 Ejercicios.....                                   | 142 |
| 7. EXCEPCIONES EN JAVA.....                           | 147 |
| 7.1. Qué es una excepción.....                        | 147 |
| 7.2. Tipos de excepciones.....                        | 147 |
| 7.3. Gestión de excepción: try...catch...finally..... | 148 |
| BIBLIOGRAFÍA.....                                     | 152 |



**LIBRO JAVA BÁSICO PARA APRENDICES**

ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hf6e-nj87>





## INTRODUCCIÓN A JAVA



*“Bien, Java podría ser un buen ejemplo de cómo debería ser un lenguaje de programación. Sin embargo, las aplicaciones Java son buenos ejemplos de cómo las aplicaciones no deberían ser.”*

*Pixadel*

En este primer capítulo se presentará la historia de Java y cuáles son sus principales características, cuáles son las herramientas de desarrollo para programar en Java, se presentará al entorno Java, **El Java Development Kit** y escribirás y compilarás tu primer programa en Java. El capítulo se cierra con unos consejos de estilo sobre la codificación en Java.

## 1.1. Historia

El lenguaje de programación Java fue desarrollado por Java son James Gosling (emac) y Bill Joy (Sun) en el año de 1991. Es un lenguaje de programación de propósito general, de objetos, que proviene del lenguaje de programación llamado Oak, cuyo objetivo fue la creación de software para la televisión interactiva.

Este proyecto (televisión interactiva) fue un completo fracaso y la meta de los creadores, se dirigió a la plataforma de Internet con el lema “La red es la computadora”.

La primera versión del lenguaje de programación Java fue publicada por Sun Microsystems en 1995.

Los criterios de diseño del lenguaje de programación Java inicial fueron:

- Independiente de la máquina.
- Seguro para trabajar en red.
- Potente para sustituir código nativo.

## 1.2. Definición de java

El lenguaje de programación java es propósito general, de objetos y es una herramienta tecnológica que, como todos los lenguajes de programación, nos sirven para escribir programas para computadora. Estos programas de computadora son de dos tipos:

- Aplicaciones – Son programas para computadora tradicionales, es decir se ejecutan únicamente en un entorno computadora.
- Applets – Son pequeños programas que se ejecutan dentro de una página Web. Para verlos hace falta un browser como Internet Explorer o Firefox.

## 1.3. Características del lenguaje Java

El lenguaje de programación Java tiene las características siguientes:

**Lenguaje de Objetos.** Dado que el lenguaje de programación java nació como un lenguaje de objetos, tiene las siguientes características inherentes a esta filosofía, como son: la abstracción, encapsulación, herencia, y polimorfismo.

**Bibliotecas disponibles.** El lenguaje de programación java, adicionalmente al juego de instrucciones propias del lenguaje y que dan soporte tecnológico en la programación de aplicaciones para computadora, provee a los programadores un amplio conjunto de clases que hace posible que se desarrolle cualquier tipo de software.

**Programación simple.** Java es un lenguaje cuyo aprendizaje es realmente sencillo y rápido, si bien es cierto que constituye una nueva forma de pensar para su aplicación en la programación resulta relativamente sencillo escribir applets interesantes desde el inicio. Los desarrolladores familiarizados con el lenguaje de programación C++ encontrarán que programar con Java en cualquiera de sus plataformas es más sencillo. Los programadores experimentados en C++ pueden migrar muy rápidamente a Java y ser productivos en poco tiempo.

**Aplicaciones Distribuidas.** Java proporciona una colección de bibliotecas clases para su uso en aplicaciones de red, que permiten abrir sockets y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas.

**Seguro.** El código Java pasa muchos tests antes de ejecutarse en una máquina. El código se pasa a través de un verificador de bytecodes que comprueba el formato de los fragmentos de código y aplica un probador de teoremas para detectar fragmentos de código ilegal —código que falsea punteros, viola derechos de acceso sobre objetos o intenta cambiar el tipo o clase de un objeto—.

Si los bytecode pasan la verificación sin generar ningún mensaje de error, entonces sabemos que:

El código no produce desbordamiento de operandos en la pila

El tipo de los parámetros de todos los códigos de operación son conocidos y correctos

No ha ocurrido ninguna conversión ilegal de datos, tal como convertir en puntero

El acceso a los campos de un objeto se sabe que es legal: public, private, protected

No hay ningún intento de violar las reglas de acceso y seguridad establecidas

Evitamos saltos a mitad de una instrucción, o direccionamientos de memoria de un objeto fuera de los límites de este.

El cargador de clases también ayuda a Java a mantener su seguridad, separando el espacio de nombres del sistema de ficheros local, del de los recursos procedentes de la red. Esto limita cualquier aplicación del tipo Caballo de Troya, ya que las clases se buscan primero entre las locales y luego entre las procedentes del exterior.

**Portable.** El hecho que el lenguaje de programación java no se identifique con ninguna arquitectura propietaria, sólo representa una parte de su portabilidad. Cuando se construye un programa de aplicación en el lenguaje de programación java, es el lenguaje quien especifica los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, por ello los programas de aplicación desarrollados en alguna plataforma es igual para cualquier plataforma. a esto se le conoce como la Máquina Virtual Java (JVM).

**Alto rendimiento.** La ejecución de un programa desarrollado en una plataforma Java alcanzan grandes velocidades por ahorrarse líneas de código lo cual lo convierte en un programa con alto rendimiento.

**Multihebra.** El lenguaje de programación Java sincroniza la ejecución de múltiples hilos (multithreading), lo que hace posible el procesamiento paralelo es decir la ejecución de varios programas (procesos) en forma simultánea, es por ello que mientras un programa está actualizando la información de un cliente, otro está calculando alguna función y otro puede estar presentando dibujos animados, esta característica tan importante facilita la implementación de aplicaciones distribuidas.

#### 1.4. Herramientas de desarrollo

El Java Development Kit (JDK) es una herramienta de desarrollo que incluye Java Runtime Environment, el compilador Java y las API de Java.

#### 1.5. Entorno Java para desarrollo

Existen varias plataformas IDEs para Java. Como, por ejemplo:

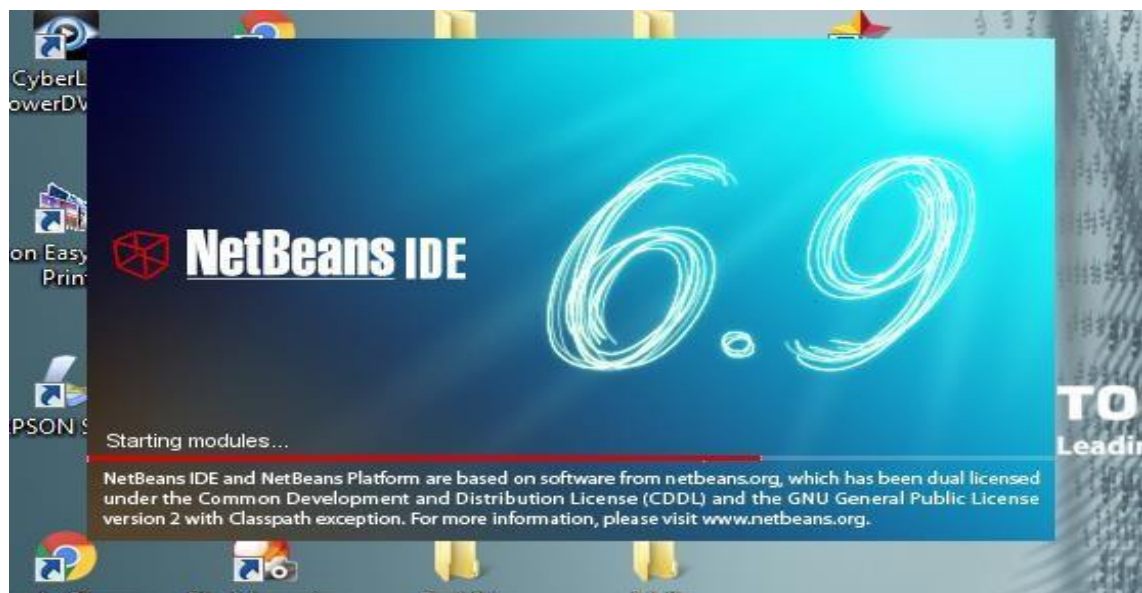
- a) **Eclipse.** Es una plataforma Java, muy usada en la actualidad y se instala como un software básico cuyas funcionalidades son expandibles en La medida que se

requieran mediante la instalación de un software que garantiza la visualización del contenido en Internet que no está diseñado para que lo procese Firefox. Es un software libre que se puede descargar en <http://www.eclipse.org>

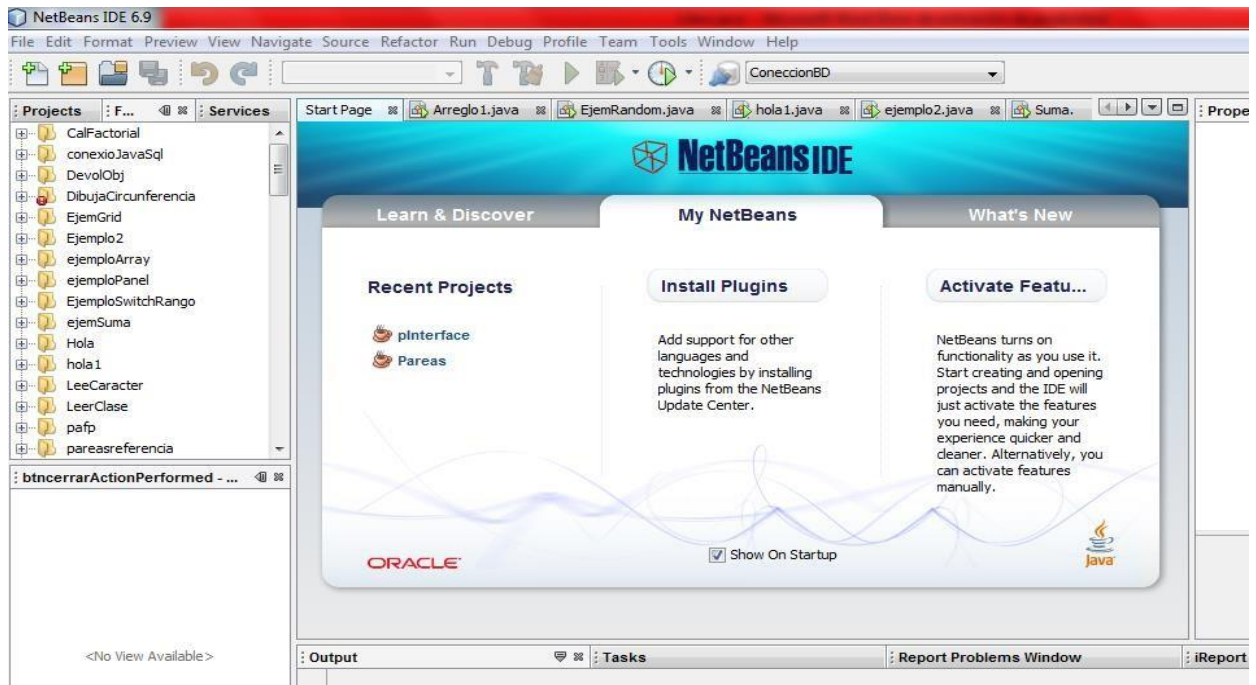
- b) **NetBeans.** Es una plataforma Java, también muy usada en la actualidad y se instala como un software básico cuyas funcionalidades son expandibles en la medida que se requieran mediante la instalación de un software que garantiza la visualización del contenido en Internet que no está diseñado para que lo procese Firefox. Es un software libre que se puede descargar en <http://www.netbeans.org>

## 1.6. El típico primer programa

En el presente trabajo se empleará la plataforma tecnológica java netBeans IDE versión 6.9, el mismo que luego de bajarlo de internet se instaló dando como resultado la siguiente imagen al momento de su ejecución.



y después de algunos segundos de espera, aparece en su pantalla la siguiente imagen;

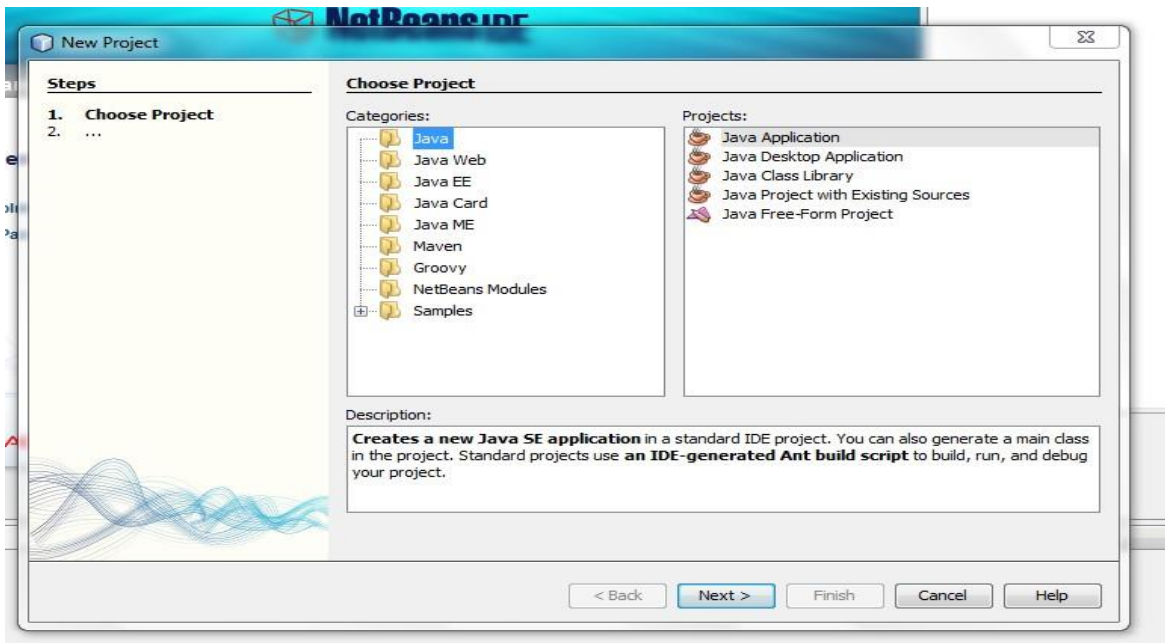


Para escribir mi primer programa que coloque en su pantalla el mensaje “HOLA MUNDO”, se realizará dos actividades en forma obligatoria, la primera se refiere a crear el proyecto y la segunda la aplicación responsable de colocar en su pantalla el mensaje deseado.

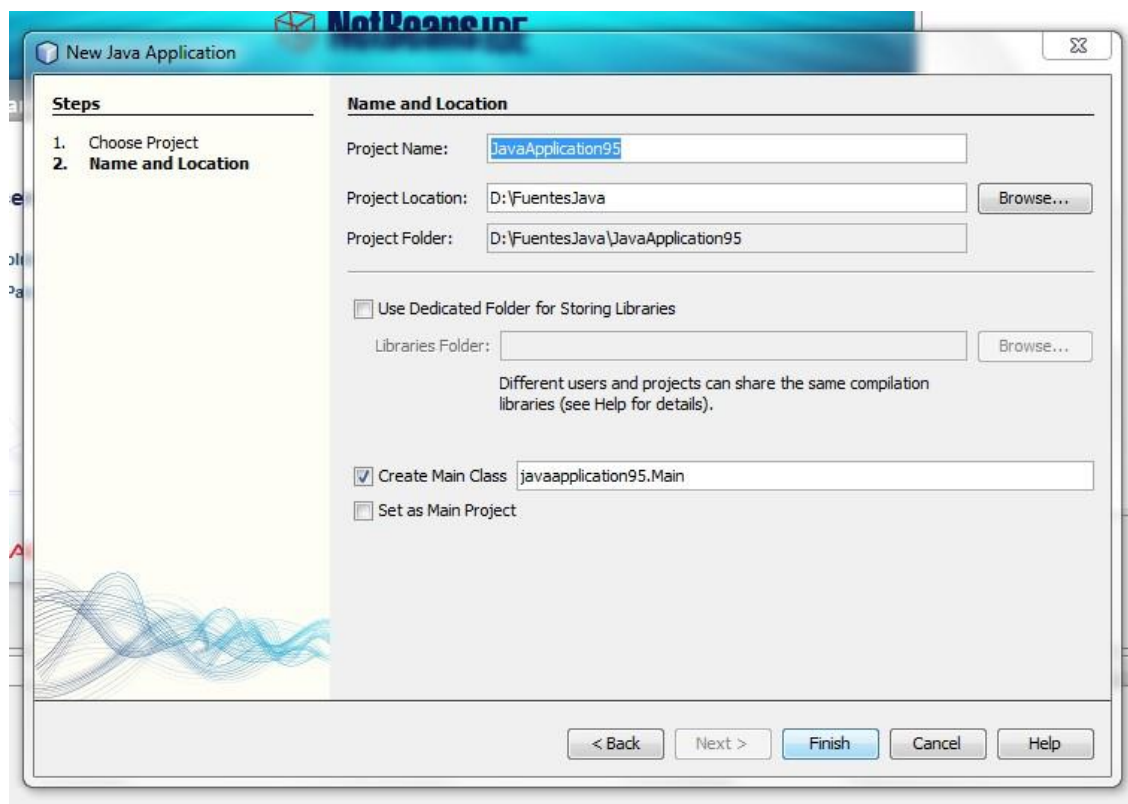
Para crear el proyecto, se ejecutarán los siguientes pasos:

1. Clic Izquierdo del mouse en la pestaña **File**
2. Seleccionar la opción **New Project**

Colocará en su pantalla la siguiente imagen:

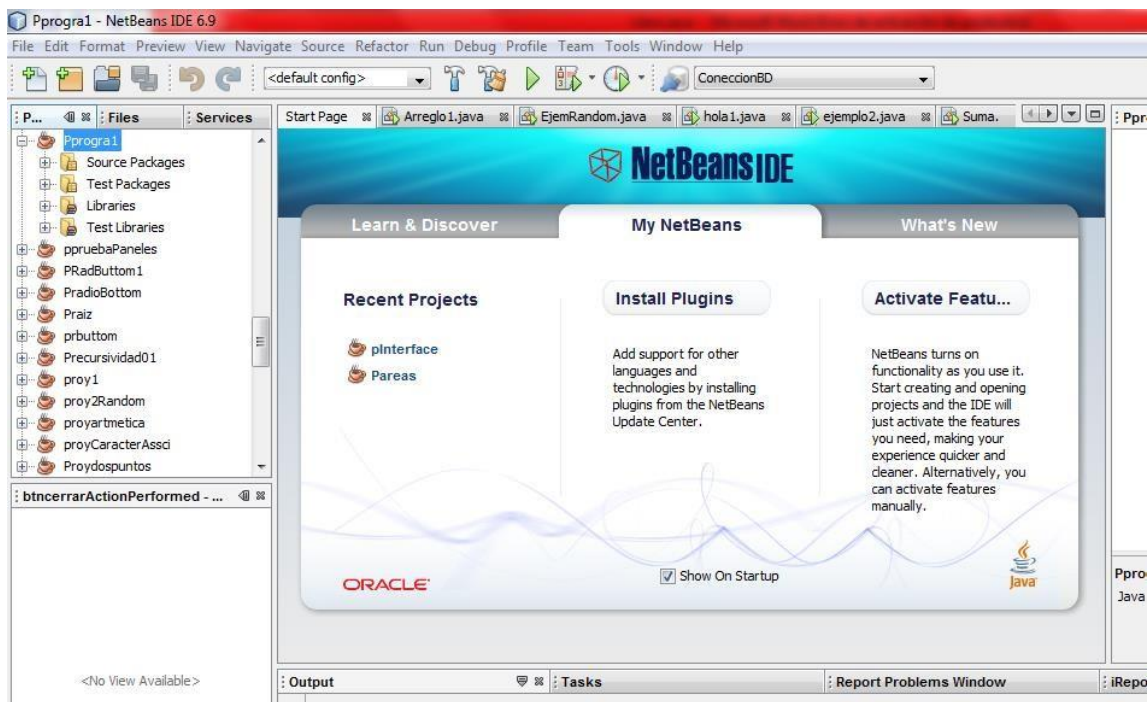


3. En esta imagen seleccionar en categoría **Java** (Resaltado en color azul) y en projects **Java Application** y luego dar clic en **botón Next**
4. La acción realizada en paso tres, colocará en su pantalla la siguiente imagen:



- En esta imagen se registrará la siguiente información:
- En Project Name: Ingresar el nombre del proyecto en este caso **Pprogra1**
- En Project Location ingresar el nombre del directorio (toda la ruta) donde se quiere guardar el proyecto creado en este caso **D:\FuentesJava**
- Fijarse que las dos casillas finales se encuentren desactivadas
- Dar clic en el botón **Finish**

5. La acción del paso 4 colocará en su pantalla la siguiente imagen:

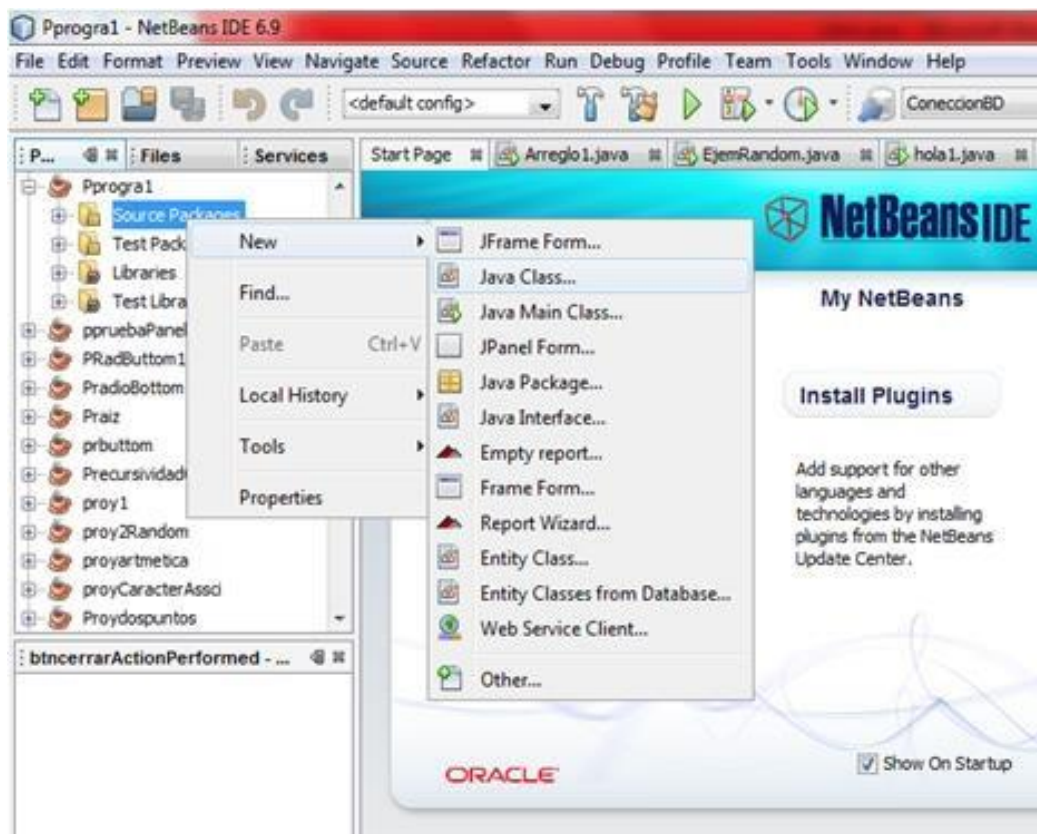


En esta imagen se puede notar lo siguiente:

1. Se ha creado el proyecto de nombre **Pprogra1**, cuyo nombre se encuentra resaltado de azul al lado superior izquierdo de su imagen
2. Se procede a escribir la primera clase HolaMundo.Java, para ello se dará clic Izquierdo en **Source Package**

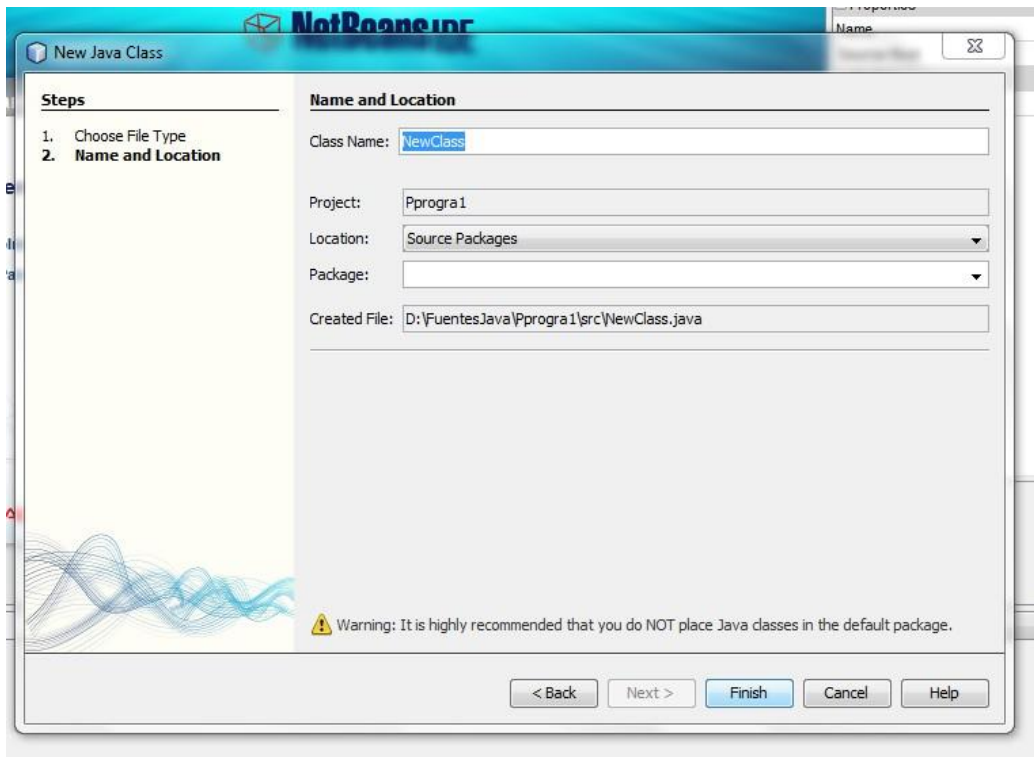


6. La acción del paso 5.2 anterior y después de seleccionar la opción **File** seguido de la opción **New File** colocará en su pantalla la siguiente imagen:



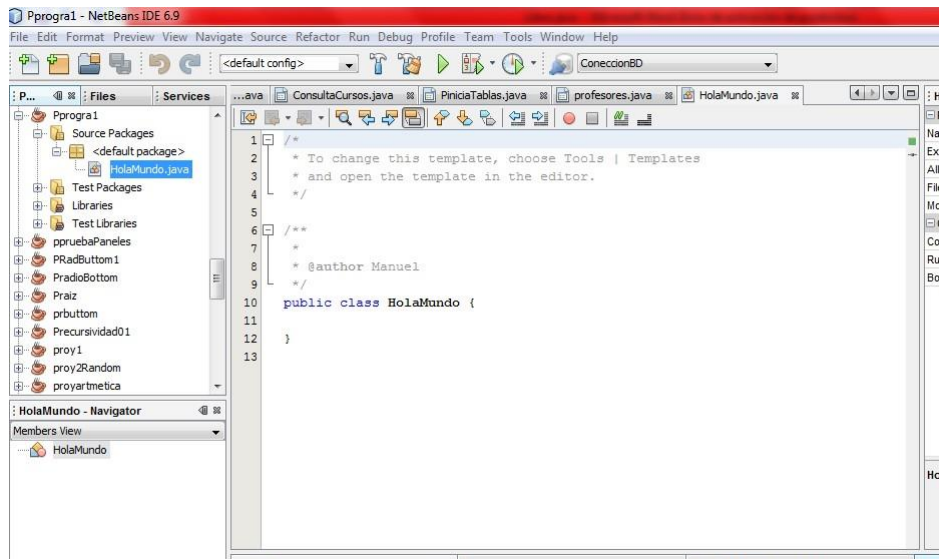
De la lista de opciones que aparecen en su imagen del lado izquierdo dar clic en **Java** y del lado derecho hacer clic en la opción **Java MainClass**

7. La acción del paso seis, colocará en su pantalla la siguiente imagen:



Donde se dará el nombre de la clase a construir: Class Name: **HolaMundo** y luego dar clic en el botón Finish.

8. La acción del paso anterior colocará en su pantalla la siguiente imagen:



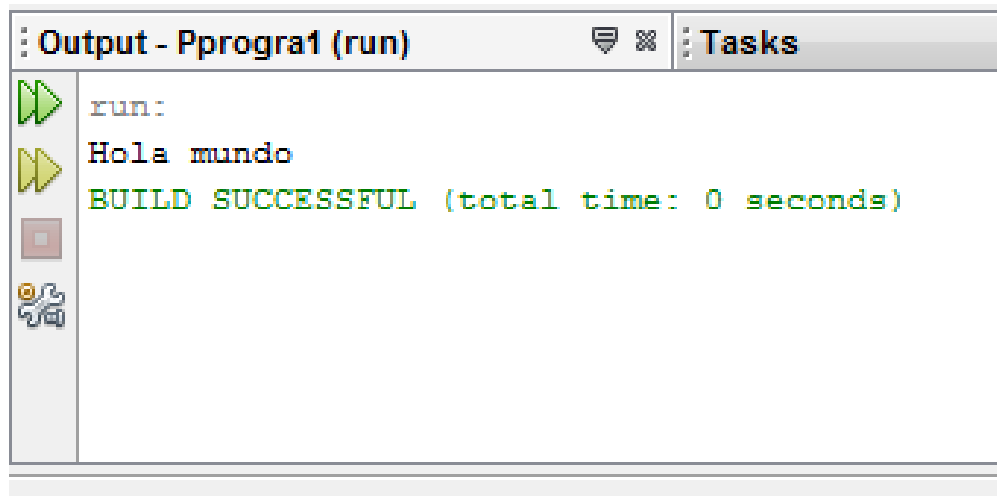
### Código inicial:

```
public class HolaMundo {  
  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
        System.out.println("Hola mundo");  
    }  
  
}
```

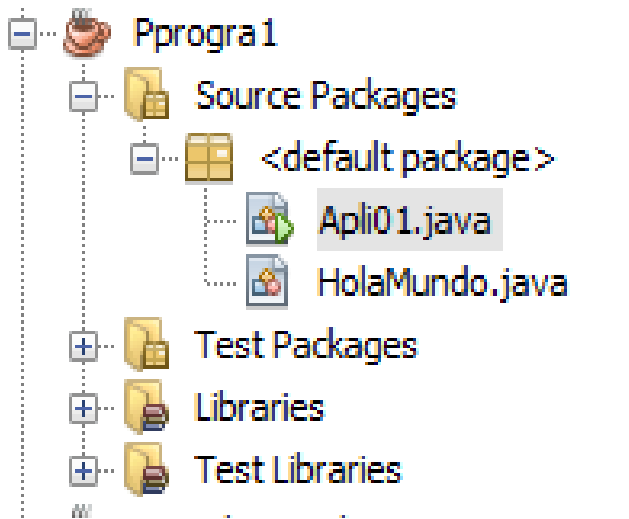
Es en este momento que se ingresará el código resaltado en amarillo en el lugar indicado de la primera clase que reporte en su pantalla el mensaje “**Hola mundo**”

9. Después de digitar el código de la aplicación en esta imagen se tiene el siguiente resultado:


Al ejecutar Pprogra1.java, se mostrará en la consola la siguiente imagen:



En la parte izquierda de su pantalla se presenta un esquema del navegador del proyecto que tiene la siguiente estructura:



y donde se aprecia lo siguiente:

Nombre del proyecto desarrollado: Pprogra1 donde **HolaMundo.java**, es un programa java y tiene la siguiente particularidad, HolaMundo.java tiene un icono  donde se aprecia una flecha de color verde y que indica el código de la clase HolaMundo.java es ejecutable, si el icono de la clase Hola mundo no presentará dicha flecha de color verde, indicando que dicho código no es ejecutable en forma directa sino a través de alguna aplicación java.

Algunas otras consideraciones, que se concluyen de la experiencia de escribir este primer programa son:

Comenzaré analizando el código de la clase HolaMundo que es como sigue:

```
public class HolaMundo {  
  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
        System.out.println("Hola mundo");  
    }  
}
```

El lenguaje de programación Java discrimina el uso de caracteres en mayúscula de los caracteres en minúscula, de allí se puede inferir que el nombre de la clase pública `HolaMundo` haría referencia a otra clase si el nombre de la clase se escribe como `holaMundo`. Cuando se escriba el nombre de una clase esta debe empezar con un carácter en mayúscula como `Hola` y si es nombre compuesto como este caso `Mundo` sigue la misma regla.

El fin de una instrucción Java se indica por el separador; como se aprecia en la siguiente instrucción:

```
System.out.println("Hola mundo");
```

Por último, se observa el uso de una llave que abre: { y a final de método una llave que cierra: }, estas llaves que en el código mostrado están resaltados de color celeste define el ámbito del método y define un bloque de instrucciones de código Java.

De la misma forma, se observa que el nombre de la clase también tiene una llave que abre: { y a final de la clase una llave que cierra: }, estas llaves que en el código mostrado están resaltados de amarillo define el ámbito de la clase y define un bloque de instrucciones de código Java como sigue:

```
public class HolaMundo {  
  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
        System.out.println("Hola mundo");  
    }  
  
}
```

## 2. SINTAXIS EN PROGRAMAS JAVA

```
1 import java.util.Scanner;
2 public class EjecutaFactorial {
3     public static void main(String[] args) {
4         // atributos
5         int n;
6         // establecer objeto de entrada
7         Scanner teclado = new Scanner(System.in);
8         // Instancia de clase
9         Factorial objFactorial = new Factorial();
10        System.out.print ("Ingrese Numero; ");
11        n=teclado.nextInt();
12        objFactorial.EstablecerNumero(n);
13        objFactorial.CalcularFactorial();
14        System.out.println("***** RESULTADOS *****");
15        System.out.println("El factorial es: "+objFactorial.ObtenerFactorial());
16    }
17 }
18
19 }
20
```

La sintaxis en programas Java, nos encausa a seguir los principios y normas al momento de construir una instrucción y todas las instrucciones Java, de un programa de una clase, cuyo objetivo es dar solución a una necesidad, ejemplo en el código mostrado en la parte superior esta clase tiene 15 instrucciones que en su conjunto tienen como fin calcular el factorial de un número.

Puntualizando la forma de colocar comentarios, declarar variables y constantes, uso de tipo de datos, uso de identificadores, enumeración de las palabreas reservadas Java, y el uso de separadores.

## 2.1. Comentarios

Los comentarios en el lenguaje de programación java sirven para etiquetar mensajes, no son ejecutables y son de tres tipos, comentarios de una línea, comentarios de bloques y comentarios de documentación.

- Comentarios de una solo línea

Empieza con los caracteres // y el enunciado del comentario

Ejemplo:

```
// Atributos
int a,b,c; // a,b,c son atributos de clase enteros
```

- Comentarios de bloques

Empieza con los caracteres iniciales /\* en cualquier parte de la línea de instrucción, luego el bloque de mensajes del comentario y finaliza con los caracteres \*/

Ejemplo:

```
/*
    Texto de los cometarios en bloque
*/
Este es un ejemplo de comentarios en bloque
/*
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
*/
```

- Comentarios de documentación

Sirve para documentar el programa. Empieza con los caracteres iniciales `/**` en cualquier parte de la línea de instrucción, luego el bloque de mensajes del comentario y finaliza con los caracteres `*/`

Ejemplo:

```
/**
Texto de los comentarios de documentación
*/

Este es un ejemplo de comentarios de documentación:
/**
*
* @author Manuel
*/
```

## 2.2 Tokens

Los tokens en el lenguaje de programación java es el elemento más pequeño de un programa java que tiene sentido para el compilador, tienen las siguientes categorías: Identificadores, palabras reservadas, operadores y separadores.

### 2.2.1. Identificadores

Los Identificadores en el lenguaje de programación java son tokens que permiten dar nombre a las variables, métodos, objetos y clases estos nombres serán únicos para el compilador y programador, se tendrá especial cuidado en respetar que estos nombres deben empezar con una letra, un carácter de subrayado «\_» o el símbolo de dolar «\$» y que java hace discriminación entre mayúsculas y minúsculas.

Ejemplos cuando empiezan con una letra: Las clases: Clase o MiClase.

Las interfaces: Interfaz o MiInterfaz.



Los métodos: `metodo ()` o `metodoLargo()`.

Las variables: `altura` o `alturaMedia`.

Las constantes: `CONSTATE` o `CONSTANTE_LARGA`.

Los paquetes: `java.paquete.subpaquete`.

### 2.2.2. Palabras reservadas

El conjunto de palabras propias del lenguaje Java las que realizan una única función dentro del programa, no pueden ser usadas como identificadores dentro de un programa de allí su carácter de reservadas en el lenguaje de programación Java, son las que se muestran a continuación:

`abstract, boolean, break, byte, byvalue, case, cast, catch, char, class, const, continue, default, do, double, else, extends, false, final, finally, float, for, future, generic, goto, if, implements, import, inner, instanceof, int, interface, long, native, new, null, operator, outer, package, private, protected, public, rest, return, short, static, super, switch, sincroniced, this, throw, throws, transient, true, try, var, void, volatile, while.`

### 2.2.3 Separadores Los paréntesis ():

Los paréntesis tienen dos connotaciones en un programa java que son las siguientes:

- Delimitan listas de parámetros en los métodos.
- Modifican la precedencia de una expresión aritmética.

#### Las llaves {}:

Las llaves en el lenguaje de programación Java se usan en los siguientes casos:

- Define un bloque de código, pueden ser anidados.
- Declaran valores iniciales de los arrays, separados por comas.

#### Los corchetes []:

Los corchetes se usan en un programa Java para:

- Declaran arrays (vectores)
- Acceder a los elementos de los arrays.

### **El punto y coma «;»:**

El punto y coma indica siempre fin de una instrucción en un programa Java.

- Termina una línea de instrucción.

### **La coma «,»:**

La coma es usada para:

- Separan identificadores en declaraciones.

### **El punto «.»:**

El punto en un programa Java se utiliza para:

- Nombrar métodos de una clase instanciada.
- Nombrar un subpaquete de un paquete.

## **2.2.4 Operadores Java**

### **2.2.4.1 Operadores de asignación Java operador**

= para Asignación simple,

ejemplo:

```
C = 10;
```

**operador +=** para Asignación suma,

ejemplo:

```
n+=3;
```

```
n=n+3;
```

Estos dos ejemplos dan el mismo resultado

**operador -=** para Asignación Resta,

ejemplo:

```
n-=3;
```

```
n=n-3;
```

Estos dos ejemplos dan el mismo resultado

**operador \*=** para Asignación Multiplicación,

ejemplo:

```
n*=3;
```

```
n=n*3;
```

Estos dos ejemplos dan el mismo resultado

**operador /=** para Asignación División,

ejemplo:

```
n/=3;
```

```
n= n/3;
```

Estos dos ejemplos dan el mismo resultado

**operador %=** para Asignación Módulo,

ejemplo:

```
n%=3;
```

```
n=n%3;
```

Estos dos ejemplos dan el mismo resultado

### 2.2.4.2 Operadores aritméticos Java

**operador** + para Suma, ejemplo  $c=a+b$

**operador** - para Resta, ejemplo  $c=a-b$

**operador** \* para Multiplicación, ejemplo  $c=a*b$

**operador** / para División, ejemplo  $c=a/b$

**operador** % para Modulo (resto), ejemplo  $c=a\%b$

**operador** ++ para Incrementar, ejemplo  $a++$  ; significa  $a=a+1$

**operador** -- para Decrementar, ejemplo  $a--$ ; significa  $a=a-1$

### 2.2.4.3 Operadores relacionales Java

**operador** < para Menor a

**operador** > para Mayor a

**operador** <= para Menor o igual a

**operador** >= para Mayor o igual a

**operador** == para Igual a

**operador** != para Diferente a

### 2.2.4.4 Operadores lógicos Java

**operador** ! para No (Complemento)

**operador** && para y (And)

**operador** || para o (or)

## 2.3. Ejercicios

1. Escribir un comentario de línea para indicar:
  - a. Para indicar zona de atributos
  - b. Para indicar zona de métodos

- c. El nombre de una relación
  - d. Hacer una aclaración en una instrucción.
2. Escribir un comentario de bloque en la cabecera de programa que indique:
    - Nombre de programa
    - Fecha de redacción
    - Función del programa
    - Autor
  3. Escribir un comentario de documentación en la cabecera de programa que indique:
    - Nombre de programa
    - Fecha de redacción
    - Función del programa
    - Autor
  4. Defina que es un token, de tres ejemplos
  5. Escriba el equivalente de las siguientes operaciones.
    - $a = a + 5$
    - $b = b - 10$
    - $c = c * 4$
    - $d = d / 6$
    - $e = e \% 10$

### 3. TIPOS DE DATOS EN JAVA

#### 3.1. Tipos de datos primitivos.

Los tipos de datos que se manejan en el lenguaje de programación Java y que se conocen como tipo de datos primitivos son para números enteros son byte, short, int y long, para números reales son float y double para campos lógicos es el boolean y para caracteres el char, las longitudes de los datos primitivos son definidos en la plataforma java. En la Tabla 3-1 se muestran estos tamaños.

**Tabla 3-1**

*Los tipos de datos primitivos en Java*

| tipo   | primitiva | definición  |
|--------|-----------|---|
| Entero | byte      | Entero en complemento a dos   |
|        | short     | con signo de 8 bits<br>Entero en complemento a dos                                |
|        | Int       | con signo de 16 bits<br>Entero en complemento a dos                               |
|        | long      | con signo de 32 bits<br>Entero en complemento a dos                               |
| Real   | float     | con signo de 64 bits<br>Real en punto flotante según la norma IEEE 754 de 32 bits |
|        | double    | Real en punto flotante según  |

Es posible recubrir los tipos primitivos para tratarlos como cualquier otro objeto en Java. Así por ejemplo existe una clase envoltura del tipo primitivo llamado Integer. La

utilidad de estas clases envoltura quedará clara cuando veamos las clases contenedoras de datos.

## Ejemplos de declaración de variables

La declaración de variables en un programa Java sigue el siguiente formato:

tipo variables;

**int a;** (Declara la variable entera a)

**int a, b;** (Declara las variables enteras a y b)

**int a, b, c;** (Declara las variables enteras a, b y c)

**int a=0;** (Declara la variable entera y la inicializa con valor cero)

**float a=0.0f, b=5.0f;** (Declara ñas variables reales a y b y las inicializa a la variable a en 0.0 y a la variable b en 5.0)

**Nota:** en el caso de solo las variables float, cuando se inicializa la variable con un valor real este valor debe terminar en f de float.

**doublé a=0.0, b=0.0;** (Declara las variables reales a y b y las inicializa con valor cero, no requiere de ninguna letra en el valor cero)

**boolean sw;** (Declara la variable lógica sw)

**booloen sw=false;** (Decakara la variable lógica sw y la iniciliaza con valor falso, las variables lógica puene etentr solamente los valores false /falso) y true (Verdad))

**char letra;** (declara la variable de tipo carácter con nombre letra)

**char letra = 'a';** (Declara la variable char letra y la inicializa con el valor de a minúscula, observe que el valor a està entre apóstrofes)

### 3.1.1. Conversión de tipos

La conversión de un tipo de datos a otro tipo es frecuente en la programación Java y se realiza de acuerdo con el requerimiento del procesamiento de datos, así se pueden tener los siguientes casos:

Veamos la conversión de un dato tipo cadena a un dato tipo entero y tipo real, inicialmente se tiene la siguiente instrucción de programa donde se declara un tipo de datos String (cadena). que no es un tipo primitivo de datos, String viene a ser una clase que se usa para declarar variables alfanuméricas y su formato de declaración es como sigue:

**String cadena = "100";** (Declara la variable de nombre cadena de tipo String y la inicializa con un valor alfanumérico 100, note que el valor 100 esta entre comillas).

Para convertir el valor de cadena a un campo numérico se emplean las siguientes instrucciones:

```
float af;
```

```
int bi;
```

```
String cadena = "100";
```

```
af = Float.parseFloat(cadena);
```

```
// ó
```

```
bi = Integer.parseInt(cadena);
```

Como pueden ver para pasar a tipo float se usa la clase Float con el método parseFloat o para pasar al tipo Integer se usa la clase Integer con el método parseInt y luego le pasamos el dato a la variable entera o float según corresponda.

En el caso anterior, debemos observar que el string declarado tiene como valor alfanumérico a "100", es decir su conformación esta dado por los dígitos 1 y 0, que son dígitos numéricos, y por ello la conversión se realiza con éxito, en caso contrario, si el valor de la cadena contiene al menos un carácter no numérico, la conversión no se realiza con éxito, y provoca en error.



Otro caso de conversión es cuando por alguna razón, se pretende pasar el tipo de datos entero al tipo de datos string o al tipo de datos float, en este caso se logra con las siguientes instrucciones de programa:

```
int a = 100;
```

```
String cadena;
```

```
Float b;
```

```
cadena = String.valueOf(a);
```

```
b = Float.parseFloat(cadena);
```

### 3.2. Tipos de datos referencia.

En Java los objetos (instancias de clases) se manejan a través de referencias. Cuando se crea una nueva instancia de una clase con el operador new este devuelve una referencia al tipo de la clase. Para aclararlo veamos un ejemplo:

Tenemos definido y programado una clase de nombre factorial como se muestra a continuación:

```
public class Factorial {  
    // Atributos  
    int n, fact;  
    // Metodos  
    public void EstablecerNumero(int n1)  
    {  
        n= n1;  
    }  
    public void CalcularFactorial()  
    {  
        fact = 1;  
        for (int i = 1; i <= n; i++)  
        {  
            fact = fact * i;  
        }  
    }  
    public int ObtenerFactorial()  
    {  
        return fact;  
    }  
}
```

```
}  
}
```

Y en la clase de aplicación, se tiene el siguiente código de programa:

```
import java.util.Scanner;  
public class EjecutaFactorial {  
    public static void main(String[] args) {  
        // atributos  
        int n;  
        // establecer objeto de entrada  
        Scanner teclado = new Scanner(System.in);  
        // Instancia de clase  
        Factorial objFactorial = new Factorial();  
        System.out.print ("Ingrese Numero; ");  
        n=teclado.NextInt();  
        objFactorial.EstablecerNumero(n);  
        objFactorial.CalcularFactorial();  
        System.out.println("***** RESULTADOS *****");  
        System.out.println("El factorial es: "+objFactorial.ObtenerFactorial());  
    }  
}
```

Podemos ubicar la instrucción:

```
Factorial objFactorial = new Factorial();
```

El operador new() reserva espacio en memoria para contener un objeto del tipo Factorial y devuelve una referencia que se asigna a objFactorial. A partir de aquí accedemos al objeto a través de su referencia. Es posible, por tanto, tener varias referencias al mismo objeto. Presta atención al siguiente fragmento de código.

```
objFactorial.EstablecerNumero(n);  
objFactorial.CalcularFactorial();  
System.out.println("***** RESULTADOS *****");  
System.out.println("El factorial es: "+objFactorial.ObtenerFactorial());
```

Si se ingresa en el programa, cuando solicita “Ingrese Numero” al valor 3, el resultado del factorial de tres, se mostrará en la salida por pantalla es:

```
***** RESULTADOS *****
```

```
El factorial es: 6
```

Como las tres referencias hacen referencia a la misma instancia, los cambios sobre el objeto se pueden realizar a través de cualquiera de ellas.

### 3.3. Recolector de basura.

Los objetos que dejan de estar referenciados a través de alguna variable no se pueden volver a recuperar. Para que estos objetos desreferenciados no ocupen memoria, un recolector de basura se encarga de «destruirlos» y liberar la memoria que estaban ocupando. Por lo tanto, para «destruir» un objeto basta con asignar a su variable referencia el valor null como puedes ver en el siguiente ejemplo:

```
public class Factorial {  
    // Atributos  
    int n, fact;  
    // Metodos  
    public Factorial(int n1)  
    {  
        n= n1;  
    }  
    public void CalcularFactorial()  
    {  
        fact = 1;  
        for (int i = 1; i <= n; i++)  
        {  
            fact = fact * i;  
        }  
    }  
    public int ObtenerFactorial()  
    {  
        return fact;  
    }  
}
```

Analizar las siguientes instrucciones de programa:

```
Factorial objFactorial = new Factorial(3);  
Factorial otroObjFactorial = new Factorial(4);  
objFactorial = new Factorial(5); // El Factorial (3) se pierde  
otroObjFactorial = null; // El Factorial (4) se pierde
```

### 3.4. Declaración de variables.

Convenciones En la Sección 3.2 se mostraron algunos ejemplos de declaraciones de variables. Al elegir su nombre recuerda seguir las convenciones que se dieron en la Sección 1.5

Siempre es aconsejable asignar un valor por defecto en el momento de declaración de una variable. En algunos casos, incluso, se producirá un error durante la compilación si hemos olvidado iniciar alguna variable.

### 3.5. Ámbito de las variables.

El ámbito de las variables está determinado por el bloque de código donde se declaran y todos los bloques que estén anidados por debajo de este.

Tenemos la definición de una clase Factorial como sigue: Si se tiene una clase declarada como sigue:

```
public class Factorial {  
    // Atributos  
    int n, fact;  
    // Metodos  
    public Factorial(int n1)  
    {  
        n= n1;  
    }  
    public void CalcularFactorial()  
    {  
        fact = 1;  
        for (int i = 1; i <= n; i++)  
        {  
            fact = fact * i;  
        }  
    }  
    public int ObtenerFactorial()  
    {  
        return fact;  
    }  
}
```

Presta atención al siguiente fragmento de código del programa de aplicación:

```
{ // Bloque externo de color amarillo

int entero = 1;

Factorial objFactorial = new Factorial(3);

    { // Bloque interno de color celeste

        // Y aquí tengo el bloque interno

        int entero = 2; // Error ya está declarada

        objFactorial = new Factorial(8); // Correcto

    }

}
```

Analizando el código de esta porción de programa se puntualiza lo siguiente:

dentro del bloque externo (llaves de color amarillo) se declara la variable entera entero, esta variable tendrá un ámbito válido para este bloque y sus bloques internos anidados de llaves de color celeste, si dentro de este bloque interno se declara la variable entero el editor del Java colocará un mensaje de error porque la variable entero está vigente y ya no puede volverse a declarar.

El ejemplo siguiente muestra en forma clara este problema:

```
{
    int entero=0;
    {
        float entero=0,0f;
    }
}
```

Cuando se escribe la instrucción **float entero=0,0f;** el compilador reportata el error de variable ya definida. a continuación, por fines didácticos se muestra un programa ejemplo en Java con la declaración de variables.

```

import java.util.Scanner;
public class Suma {
    public static void main(String[] args) {
        short as;
        int a,b,c;
        float x=0.0f, y=0.0f, z=0.0f;
        Scanner teclado = new Scanner(System.in);
        System.out.print("Ingrese Num. A: ");
        a= teclado.nextInt();
        System.out.print("Ingrese num B: ");
        b=teclado.nextInt();
        c=a + b;
        System.out.println("***** RESULTADOS *****");
        System.out.println("La valor de la Suma es : "+c);
        System.out.print("Ingrese el valor de x: ");
        x = teclado.nextFloat(); // realiza la suma
        System.out.print("Ingrese el valor de y: ");
        y = teclado.nextFloat();
        z = x+y;
        System.out.println("***** RESULTADOS REALES FLOAT
*****");
        System.out.println("La valor de la Suma es : "+z);
        double x1=0.0, y1=0.0, z1=0.0;
        System.out.print("Ingrese el valor de x1: ");
        x1 = teclado.nextDouble();
        System.out.print("Ingrese el valor de y1: ");
        y1 = teclado.nextDouble();
        z1 = x1+y1;
        System.out.println("***** RESULTADOS REALES DOUBLE
*****");
        System.out.println("La valor de la Suma es : "+z1);
        {
            int entero=0;
            {
                float entero=0,0f;
            }
        }
    }
}

```

La declaración de variables en el programa está resaltada con amarillo, la instrucción resaltada en celeste corresponde a un error.

### 3.6 Ejercicios

1. Declare los siguientes tipos de datos:
  - entero a
  - entero x , y
  - doublé x
  - doublé x, y
  - entero a, b, c con valores iniciales 5, 6, 7 respectivamente
  - lógico sw con valor inicial falso
  - Cadena cad con valor nulo
  - Declare una constante para PI con valor 3.14
2. Convertir cadena = “200” a un entero en la variable E
3. Convertir cadena = “ 200 ” a un entero en la variable E1
4. Convertir cadena = “200” a un float en la variable fl
5. Convertir el float 23.84f a una cadena en la variable nomb
6. Defina que es un dato tipo referencia, de tres ejemplos

## 4. ESTRUCTURAS DE CONTROL EN JAVA

Las estructuras de control en Java presentan escasas diferencias con respecto a C/C++, no obstante, existen diferencias. Recordemos que se llama programación estructurada al uso correcto de las estructuras de control, que se resume en que toda estructura de control debe tener un único punto de entrada y un único punto de salida.

Las estructuras de control son fundamentales a la hora de realizar un programa ya que son las que nos dicen que comportamiento tienen los datos, existen 5 estructuras básicas que hay que aprender a manejar las cuales son el if else, switch, for, while y do while.

### 4.1. Estructuras condicionales.

Dos son las estructuras de control condicionales en Java: bifurcación y selección múltiple.

#### 4.1.1. Bifurcación: if-else, if-else-if

Su sintaxis es:

```
if(condicion) {  
    instruccion1();  
    instruccion2();  
    // etc }  
else {  
    instruccion1();  
    instruccion2();  
    // etc }
```

Es necesario que la condición sea una variable o expresión booleana. Si sólo existe una instrucción en el bloque, las llaves no son necesarias. No es necesario que existe un bloque else.

Se pueden anidar como en el siguiente ejemplo



```

if(condicion1) {
    bloqueDeInstrucciones();
}
else if(condicion2) {
    bloqueDeInstrucciones();
}
else {
    bloqueDeInstrucciones();
}

```

Ejemplo: se tiene el siguiente código:

```

if( a==4){ /* si lo que contiene la variable a es igual a 4 muestra "el numero es un
    cuatro"*/
    System.out.println("el numero es un cuatro");
}
int b=0;
int valor=a+b;
System.out.println(valor);

```

En este caso tenemos el mismo if pero sin su respectivo else, quiere decir que en caso de que 'a' sea igual a 4 se mostrara en pantalla el mensaje dado, pero en el caso que 'a' no contenga un 4 el programa continuara su flujo normal y realizara la suma de a+b que se guarda en la variable llamada "valor" y mostrara el contenido de valor en pantalla incluso si el contenido de 'a' fuese igual a 4 la suma se llevara a cabo ya que se encuentra fuera del if y el programa es interpretado de manera descendente es decir de arriba a abajo.

### **Programa Ejemplo del uso de la sentencia IF**

```

package ejemploif;
import java.util.*;
public class Main {
    public static void main(String[] args) {
        // Atributos
        int a=28, b=36, c= 14;
        boolean mayor = false;
        int may, men;
        // Calculo del mayor de tres numeros dados primer metodo
        if (a > b){
            if (a > c) System.out.println ("el numero mayor es :"+a);
            else System.out.println ("el numero mayor es :"+c);
        }
        else {

```

LIBRO JAVA BÁSICO PARA APRENDICES  
 ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hf6e-nj87>

```

        if (b > c) { System.out.println ("el numero mayor es :"+b);}
        else
            System.out.println ("el numero mayor es :"+c);
    }
// mayor y menor segundo metodo
// calculo del mayor
mayor= false;
may = a;
if (a > may){
    mayor = true;
    may = a;
}
if (b > may){
    mayor = true;
    may = a;
}
if (c > may){
    mayor = true;
    may = c;
}

if (mayor) System.out.println("EL NUMERO MAYOR ES: "+may);
}
}

```

#### 4.1.2. Selección múltiple: switch.

Su sintaxis es la siguiente:

```

switch(expresion) {
    case valor1: instrucciones();
                break;
    case valor2: instrucciones();
                break;
    default: instrucciones();
}

```

La expresión ha de ser una variable de tipo entero o una expresión de tipo entero.

Cuando se encuentra coincidencia con un case se ejecutan las instrucciones a él asociadas hasta encontrar el primer break.

Si no se encuentra ninguna coincidencia se ejecutan las instrucciones en default. La sección default es prescindible.

Ejemplo: El siguiente porción de código ilustra la utilización de la sentencia de control switch:

```
int a = 4;
switch (a) {
    case 1: System.out.println ("Es uno");
        break;
    case 2: System.out.println ("Es dos");
        break;
    case 3: System.out.println ("Es tres");
        break;
    case 4: System.out.println ("Es cuatro");
        break;
    case 5: System.out.println ("Es cinco");
        break;
    case 6: System.out.println ("Es seis");
        break;
    case 7: System.out.println ("Es siete");
        break;
    case 8: System.out.println ("Es ocho");
        break;
    case 9: System.out.println ("Es nueve");
        break;
    default : System.out.println("no esta entre 1 y 9"
}
}
```

Este código va a imprimir en su pantalla el mensaje es cuatro, porque la variable entera n ha sido inicializada con el valor cuatro, en general se puede concluir en sentido de que este programa colocara en su pantalla un mensaje que depende del valor que tenga la variable entera n antes de la sentencia de control switch, si el valor de esta variable fuera un valor mayor a 9 o menor a 1 este programa colocara en su pantalla el mensaje “no está entre 1 y 9” que corresponde a la opción default.

### **Programa ejemplo del uso de la sentencia switch**

#### **Programa de la clase:**

```
public class Switch {
// Atributos
```

*LIBRO JAVA BÁSICO PARA APRENDICES*  
ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hf6e-nj87>

```

int n;
String msj="";
// Metodos
public Switch (int n){
    this.n = n;
}
public void ejecutar(){
    switch (n) {
        case 1:
        case 2:
        case 3:
            msj = "Verano";
            break;
        case 4:
        case 5:
        case 6:
            msj = "Otoño";
            break;
        case 7:
        case 8:
        case 9:
            msj = "Invierno";
            break;
        case 10:
        case 11:
        case 12:
            msj = "primavera";
            break;
        default:
            msj = "Mes errado";
    }
}
public String mostrar(){
    return "Estacion del mes: "+msj;
}
}

```

Nota: la sentencia break incorpora un salto del control del programa al final del programa

### Programa aplicación (ejecutable)

```

import java.util.Scanner;
public class apliSwitch {
    public static void main(String[] args) {
        // Atributos

```

*LIBRO JAVA BÁSICO PARA APRENDICES*  
 ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hf6e-nj87>

```

int mes;
// Metodos
Scanner tecla = new Scanner(System.in);
System.out.print("Ingrese el mes: ");
mes = tecla.nextInt();
Switch objSw = new Switch(mes);
objSw.ejecutar();
System.out.println(objSw.mostrar());
}
}

```

El programa anterior informa la estación del año, según un mes dado.

## 4.2. Estructuras de repetición.

En Java las estructuras de repetición son las mismas que en C/C++.

A continuación, se detallan y se indican las pequeñas diferencias con respecto a las consideraciones en el lenguaje de programación C/C++.

### 4.2.1. Repetición sobre un rango determinado. for

El for es una estructura de repetición que nos permite realizar operaciones cuantas veces necesitemos. El for consta de 3 partes para funcionar: la variable de inicio, la condición y el incremento.

Bucle for, su sintaxis y formato es la siguiente:

```

for(iniciación; condición; incremento) {

// Bloque de instrucciones

}

```

No es necesario que la condición se base exclusivamente en la variable de control del bucle.

En la parte de iniciación se puede declarar una variable de control del bucle cuyo ámbito será el bucle. Tanto en la parte de iniciación como de incremento se puede incluir varias expresiones separadas por comas, pero nunca en la parte de condición.

La condición ha de ser una variable booleana o una expresión que se evalúe a un valor booleano.

Ejemplos:

```
for (int i=0;i<=5;i++){  
/* variable de inicio =i, condición i menor igual a 5,incremento i++ es igual que i+1 */  
System.out.println("hola");  
}
```

Como resultado este for imprimiría 6 veces la palabra hola ya que en java el ciclo for siempre comienza por el valor definido que en este caso es igual a 0 así que si quisiéramos 10 repeticiones la condición debería de ser  $i \leq 9$ , empezando por cero.

```
for (int j = 1; j < 4;j++){  
System.out.print(objArray01.muestra(j)+" ");  
}
```

Este código declara e inicializa con valor 1, la variable entera j, la que tendrá un ámbito solo dentro de bloque de la sentencia for.

La condición  $j < 4$ , sera evaluada al inicio del bucle, si el resultado de la evaluación es verdadera, entonces se ejecutara las instrucciones del bloque del for, en caso contrario estas instrucciones no se ejecutan y el programa continua con las instrucciones siguientes.

El incremento  $j++$  se realiza a partir del segundo bucle, en este caso el incremento es de uno en uno, en este ejemplo el for ejecuta tres bucles.

```
int suma = 0;  
for (int j = 1, int k = 1; j<= 10;j++, k+=2){  
suma = suma + k; //Esta sentencia también se puede reemplazar por suma +=k  
}  
System.out.print("\n la suma es: "+suma);
```

Este código imprime la suma de los diez primeros números impares que comienzan por uno, las variables de inicio son j y k, son declaradas e inicializadas en uno en la sentencia,

tienen una separación por coma, en este ejemplo la variable `j` es un contador que sirve para contabilizar a los diez números, la condición `j <= 10` es verdadera solamente en el caso de que se esté en el rango de los diez números impares que se quieren sumar, en el incremento se incrementa a uno el contador, se incrementa en dos los números impares, ambos incrementos están separados por coma.

#### 4.2.2. Repeticiones condicionales: while, do while. Sentencia while:

El `while` al igual que el `for` es una estructura de repetición con la única diferencia de que el `while` no requiere una variable de control ni un incremento solo requiere de una condición para funcionar el `while` analiza la condición si esta se cumple realiza la acción y vuelve a preguntar cuando la condición no se cumpla el ciclo parará y seguirá el flujo normal del programa.

Su sintaxis y funcionamiento son iguales que en `C/C++`, en la estructura de control `while` evalúa la condición antes de ejecutar el bloque de la estructura.

```
while(condición) {  
  
    // Bloque de instrucciones; }
```

#### Ejemplo:

```
int A=0;  
int B=5;  
while(A < B) { //repetir mientras que A sea menor que B//  
    A=A+1;  
}
```

**Este código ejemplo de la sentencia while se detendrá cuando A sea mayor a B, o sea cuando la variable A tenga un valor de 6, cada vez que el while hace un ciclo A se incrementa en 1 cuando A sea igual a 6 el while se detendrá.**

#### Programa ejemplo del uso de la sentencia while

```
/*  
 * To change this template, choose Tools | Templates  
 * and open the template in the editor.
```

```

*/

package ejemplowhilesuma;

/**
 *
 * @author Manuel
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        int A=0;
        int B=10;
        int suma = 0;
        while(A < B) { //repetir mientras que A sea mayor que B//
            A=A+1;
            suma +=A;
        }
        System.out.println("Suma: "+suma);
    }
}

```

### **Sentencia do while:**

La sentencia do while es una estructura de control de repetición que nos permite ejecutar al menos una vez el bloque del “do”, en caso de que se cumpla la condición del “while” que hayamos programado, el control del programa volverá al punto en el cual se declaró el "do" y volverá a ejecutar el bloque declarado del “do”, la sintaxis es la siguiente:

```

do {

    // Bloque de instrucciones

} while(condición);

```

Igual que en el caso del for la condición ha de ser una variable booleana o una expresión que se evalúe a un valor booleano.

Ejemplo:



```
int x=5;// el valor inicial de x es 5//
do {
    System.out.println("x es " + x--);//la instrucción que se repetirá//
} while ( x > 0 ); //si x mayor que 0 entonces ir a donde esta do//
```

### Ejemplo del programa completo del do while

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package ejemplodowhile;

/**
 *
 * @author Manuel
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        int x=5;// el valor inicial de x es 5//
        do {
            System.out.println("x es " + x--);//la instrucción que se repetirá//
        } while ( x > 0 ); //si x mayor que 0 entonces ir a donde esta do//

    }

}
```

La salida de este programa será:

```
x es 5
x es 4
x es 3
x es 2
x es 1
```

#### 4.2.3. Uso de break y continue.

La palabra reservada break además de para indicar el fin del bloque de instrucciones en una instrucción de selección múltiple switch, sirve para forzar la salida del bloque de una estructura de repetición.

La palabra reservada `continue`, dentro del bloque de una estructura de repetición condicional, sirve para forzar la evaluación de la condición.

Observa los dos ejemplos siguientes y la salida que proporcionan por consola:

```
public class Break {
    public static void main(String [] args) {
        for(int i = 0; i < 10; i++) {
            for(int j = 0; j < 10; j++) {
                if(j > i) break;
                System.out.print(j+",");
            }
            System.out.println("");
        }
    }
}
```

La salida de este código es:

```
0,
0,1,
0,1,2,
0,1,2,3,
0,1,2,3,4,
0,1,2,3,4,5,
0,1,2,3,4,5,6,
0,1,2,3,4,5,6,7,
0,1,2,3,4,5,6,7,8,
0,1,2,3,4,5,6,7,8,9,
```

```
public class Break {
    public static void main(String [] args) {
        for(int i = 0; i < 10; i++) {
            for(int j = 0; j < 10; j++) {
                if(j < i) continue;
                System.out.print(j+",");
            } System.out.println("");
        }
    }
}
```

La salida de este código es:

LIBRO JAVA BÁSICO PARA APRENDICES  
ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hf6e-nj87>

0,1,2,3,4,5,6,7,8,9,  
1,2,3,4,5,6,7,8,9,  
2,3,4,5,6,7,8,9,  
3,4,5,6,7,8, 9,  
4,5,6,7,8,9,  
5,6,7,8,9,  
6,7,8,9,  
7,8,9,  
8,9,  
9,

### 4.3. Recursividad.

La recursividad es una técnica de programación que permite que un bloque de instrucciones se ejecute n veces. Reemplaza en ocasiones a estructuras repetitivas.

Para que un problema pueda ser resuelto de modo recursivo ha de poseer las siguientes dos características:

El problema original se ha de poder reducir al mismo problema y con una talla menor.

Debe existir un caso base o condición de parada. Por ejemplo, la función factorial se define en los siguientes términos:

- $F(n) = n * F(n-1)$ ;
- $F(0) = 1$ ;

La primera parte de la definición de la función factorial nos dice que se puede calcular la factorial de un número multiplicando ese número por la factorial del número menos uno, se ha reducido el problema al mismo problema, pero con una talla menor.

La segunda parte de la definición corresponde al caso base, que fuerza la salida de la recursividad. Java soporta la programación recursiva. A continuación, se muestra una posible solución recursiva para el cálculo de la función factorial:

```
package pfactorial;  
import java.util.Scanner;
```

*LIBRO JAVA BÁSICO PARA APRENDICES*  
ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hj6e-nj87>

```

public class Main {
    public static void main(String[] args) {
        long n;
        Scanner teclado = new Scanner(System.in);
        System.out.print("Ingrese un numero: ");
        n=teclado.nextLong();
        System.out.println("El factorial de " + n + " es: " + Factorial(n));
    }
    public static long Factorial(long n) {
        long resultado=1;
        if(n > 0) resultado = n * Factorial(n-1);
        return resultado;
    }
}

```

Otro ejemplo de recursividad es cuando se quiere imprimir una serie de números enteros que comience por 5 hasta 1 mediante la técnica de recursividad.

El código de clase es:

```

public class recursividad01 {
    void imprimir(int x) {
        if (x>0) {
            System.out.println(x);
            imprimir(x-1);
        }
    }
}

```

El código de la aplicación es:

```

public class apliRecursividad01 {
    public static void main(String[] args) {
        recursividad01 re=new recursividad01();
        re.imprimir(5);
    }
}

```

LIBRO JAVA BÁSICO PARA APRENDICES  
 ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hf6e-nj87>

```
}  
}
```

Desde la función main se llama a la función imprimir de la clase recursividad01 y desde la aplicación se le envía el valor 5. El parámetro x en la clase recursividad01 recibe el valor 5. Se ejecuta el algoritmo de la función imprimir en la clase recursividad01, imprime el contenido del parámetro (5) y seguidamente si el valor del parámetro es mayor a 0, se llama a la función imprimir, en este caso, se llama a sí misma (por eso decimos que es una función recursiva), enviándole el valor 4.

El parámetro x recibe el valor 4 y se imprime en pantalla el cuatro, llamando nuevamente a la función imprimir enviándole el valor 3.

Si continuamos este algoritmo podremos observar que en pantalla se imprime:



```
5  
4  
3  
2  
1
```

Este proceso se repite hasta que el valor del parámetro sea cero, condición que para el programa.

Tener en cuenta que cada llamada a una función consume 4 bytes por la llamada y en este caso 4 bytes por el parámetro x que es una variable entera int.

#### 4.4 Pasos para desarrollar e implementar un programa orientado a objetos

Un programa desarrollado en Java es estructurado según los convenios indicados para cada clase según su aplicación, así la estructura de una clase será como se indica en el siguiente ejemplo:

```
public class recursividad01 {  
// Declaración de atributos  
int y;  
// Declaración de metodos  
void imprimir(int x) {  
    y = x;  
    if (y>0) {  
        System.out.println(y);  
        imprimir(y-1);  
    }  
}  
}
```

Es importante puntualizar que esta estructura tiene dos partes, la primera que se declaran después del nombre de la clase, pertenece a la declaración de variables (atributos) que son los atributos de la clase y la segunda parte es donde se declaran los métodos de clase como es en este caso el método imprimir (), este tema será tratado profundamente en el capítulo V de este libro.

La segunda clase, corresponde a la clase principal desde donde se ejecutan los métodos de las clases declaradas mediante dos operaciones fundamentales, que son: La declaración de las instancias de clase (Objetos), y la ejecución de dichos objetos. Como ejemplo de esta estructura se puede mostrar el siguiente código:

```
public class apliRecursividad01 {  
    public static void main(String[] args) {  
        recursividad01 re = new recursividad01();  
    }  
}
```

```
re.imprimir(5);  
}  
}
```

En este código se muestra la forma de declarar una instancia de clase u objeto, esto se hace con la siguiente instrucción:

```
recursividad01 re = new recursividad01();
```

**Recusividad01** es el nombre de la clase, re el nombre de objeto creado mediante la cláusula new.

```
re.imprimir(5);
```

Es la instrucción que permite ejecutar el método **imprimir()** del objeto re, y el valor establecido como 5 es un valor de un parámetro inicial, de tipo entero.

#### 4.4.1. Análisis del problema.

**Planteamiento del problema.** El planteamiento del problema, es definir en forma definitiva el texto de enunciado del problema, este texto debe ser totalmente claro y objetivo, en esta etapa se define si la solución del problema requiere del uso del computador o no, en caso de requerir el uso del computador entramos a realizar su análisis, este proceso es fundamental, y es aquí, donde se bosqueja una buena solución al problema, es decir que la solución encontrada tiene que ver con el análisis inicial que se hace en esta etapa, el análisis consiste en especificar básicamente lo siguiente:

¿Cuáles son las variables de entrada al proceso de solución?

¿Cuáles son las variables de estado del proceso de solución?

¿Cuáles son las variables de salida del proceso de solución?

En el primer caso, es importante definir el tipo de variable a utilizar, está relacionado con el dominio de dicha variable con relación al sistema de información que lo va a requerir.

En el segundo caso, es importante indicar que este tipo de variables están relacionados generalmente con los cálculos que se van a realizar, y que se definen y utilizan dentro del proceso.

Las variables de salida son las mostrar el programa después de hacer el proceso de cálculo.

Una vez que se determinan estas variables, pasamos a identificar los objetos que intervienen en la solución del problema y se define que variables o atributos pertenece o se operan en los objetos identificados, estos objetos, corresponden generalmente a los sustantivos utilizados en el enunciado del problema.

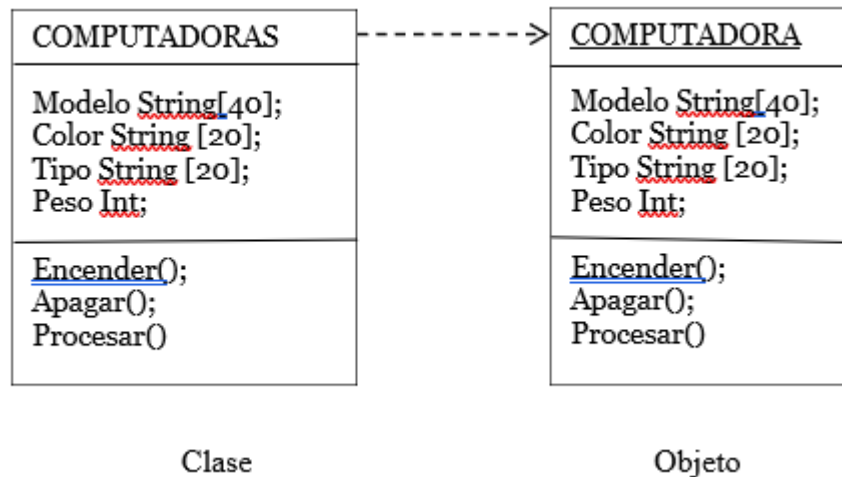
Identificar Objetos. La identificación de los objetos es realizada en esta etapa, la etapa del análisis, y para ello, se debe tener bien en claro el concepto que define a un objeto, pues un objeto viene a ser un ente, entidad, un sujeto o cosas que está en el problema, tiene un nombre único y es capaz de almacenar o procesar información.

Los objetos, encapsulan los datos y los métodos que definen su comportamiento, los datos guardan relación con las variables referidas líneas arriba, y en la teoría orientado a objetos son denominados atributos, y sus longitudes depende del tipo de dato especificado, no depende de la maquina a usar, sino del lenguaje de programación elegido.

Los métodos que definen el comportamiento del objeto procesan la información que contiene sus atributos, se representa por un nombre, el tipo de información que retorna, y la definición de los parámetros que operan.

Se representan por un rectángulo, y su nombre va subrayado.

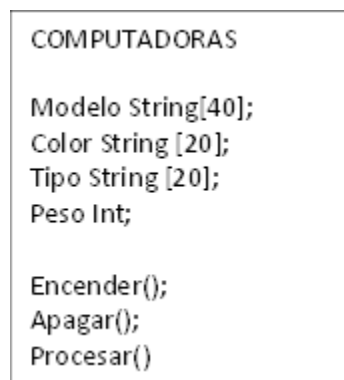




**Figura 1. Instancia de Clase (objetos). Fuente. Elaboración propia.**

#### 4.4.2. Diseño del programa.

1. Se diseña el diagrama de clases. Es el diagrama más importante en la Programación orientada a objetos (POO), sirve de base como una guía para desarrollar el código de la aplicación que representa, se representa por un rectángulo como se indica a continuación:

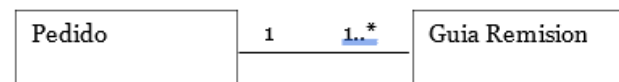
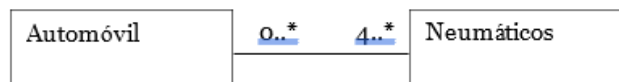
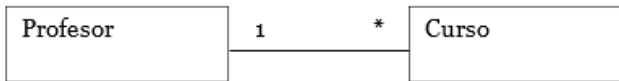
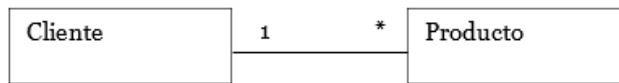


**Figura 2. Diagrama de Clase. Fuente. Elaboración propia.**

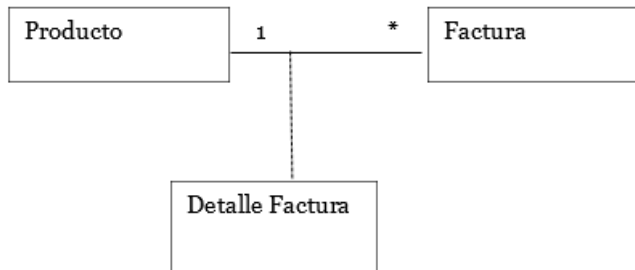
Se pueden realizar cuatro operaciones de relación con las clases, para generar un diagrama de clases, para ello es necesario saber los componentes de estos diagramas:

**Relación entre clases.** Multiplicidad: es el grado de relación que existe entre objetos de clases, ejemplo \* significa muchos

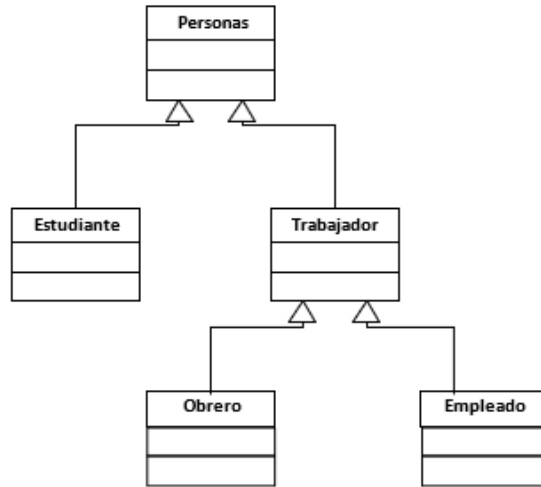
**Relación binaria:**



**Relación atributo de enlace:**



## Relación de Herencia o generalización



## Relación de Composición/Agregación

2. Se diseña la lógica de las clases en pseudocódigo

### 4.4.3. Programación.

1. Después de hacer el planteamiento del problema se hace el análisis de este
2. Se hace el diseño (algoritmos)
3. Se selecciona un lenguaje de programación
4. Se escribe programa fuente
5. Se compila y cataloga
6. Se ejecuta y prueba

### 4.4.4. Implementación.

En esta etapa se pone el programa en producción, es decir inicia la etapa de explotación, en las empresas que siguen una metodología de desarrollo de sistemas

paralelamente a esta etapa de hacer un seguimiento al funcionamiento del sistema pudiendo realizar actividades de actualización o mejora continua, que es lo más recomendable.

#### 4.5. Arrays.

Los arrays o arreglos son estructuras de datos que almacenan valores de datos del mismo tipo cuyos elementos ocupan un lugar en la memoria del computador en forma secuencial y consecutiva, a los cuales se accesa por el valor de un indicador numérico que empieza con un valor de cero para el componente uno, el valor de uno para el componente dos y así sucesivamente.

El uso de arrays en Java es distinto al uso en C/C++. En Java los arrays son objetos, instancias de la clase Array, la cual dispone de ciertos métodos útiles.

La declaración sigue la misma sintaxis que en C/C++, se debe declarar el tipo base de los elementos del array. El tipo base puede ser un tipo primitivo o un tipo referencia:

```
int arrayDeEnteros[] = null; // Declara un array de enteros
```

```
Punto arrayDePuntos[] = null; // Declara un array de referencias a Puntos
```

La creación del array se hace, como con cualquier otro objeto, mediante el uso del operador new():

```
arrayDeEnteros = new int[100]; /* Crea el array con espacio para 100 enteros */  
arrayDePuntos = new Puntos[100]; /* Crea el array con espacio para 100 referencias  
a punto */
```

En el primer caso se reserva espacio para contener 100 enteros.

En el segundo caso se crea espacio para contener 100 referencias a objetos de la clase Punto, pero no se crea cada uno de esos 100 objetos.

En el siguiente ejemplo se muestra cómo se crea cada uno de esos 100 objetos de la clase Punto y se asignan a las referencias del array.

```
for(int i = 0; i < 100; i++) arrayDePuntos[i] = new Punto();
```

Los arrays se pueden iniciar en el momento de la creación, como en el siguiente ejemplo:

```
int arrayDeEnteros[] = {1, 2, 3, 4, 5};
```

El recorrido del arreglo arrayDeEnteros en sus elementos es:

arrayDeEnteros [0] tiene valor 1

arrayDeEnteros [1] tiene valor 2

arrayDeEnteros [2] tiene valor 3

arrayDeEnteros [3] tiene valor 4

arrayDeEnteros [4] tiene valor 5

```
Punto arrayDePuntos[] = {new Punto(), new Punto(1.0f, 1.0f)};
```

Los arrays disponen de un atributo llamado length que indica el número de elementos que contiene el arreglo.

```
int NroElementos = 0;
```

```
NroElemntos = arrayDeEnteros.length;
```

Cuando se ejecute esta instrucción el valor NroElementos será de 5, que el arreglo tiene 5 elementos.

También proporcionan un método para copiar partes de un array sobre otro array:

```
System.arraycopy(origen, inicioOrigen, destino, inicioDestino, longitud);
```

La extensión a arrays de más dimensiones es directa:

```
Punto matriz[][] = new Punto[3][3]; // Declaramos una matriz
```

A continuación, se presenta el código de un programa Java que maneja arreglos:

Para la clase arreglo:

```
public class Array01 {  
// Atributos
```

*LIBRO JAVA BÁSICO PARA APRENDICES*  
ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hf6e-nj87>

```

int arrayDeEnteros1[] = {1, 2, 3, 4, 5};
int arrayDeEnteros2[] = new int[10];
int xlong;
int ocupados = 0;
// Metodos
public void Copia(){
    System.arraycopy(arrayDeEnteros1,0,arrayDeEnteros2,1,3);
}
public int muestra(int j){
    ocupados = j;
    return arrayDeEnteros2[j];
}
public int regresaLongitud2(){
    xlong = arrayDeEnteros2.length;
    return xlong;
}
public int regresaOcupados2(){
    return ocupados;
}
}

```

Para la clase aplicación (Ejecutable):

```

public class jArray01 {
    public static void main(String[] args) {
        Array01 objArray01 = new Array01();
        System.out.println("longitud inicial del arreglo2:
"+objArray01.regresaLongitud2());
        System.out.println("Ocupados inicial del arreglo2:
"+objArray01.regresaOcupados2());
        objArray01.Copia();
        for (int j = 1; j< 4;j++){
            System.out.print(objArray01.muestra(j)+" ");
        }
        System.out.println();
        System.out.println("longitud final del arreglo2:
"+objArray01.regresaLongitud2());
        System.out.println("Ocupados final del arreglo2:
"+objArray01.regresaOcupados2());
    }
}

```

Igual que en el caso de los arrays unidimensionales lo importante es saber que el hecho de declarar un array, no crea los objetos que se referenciaran desde las posiciones del array, únicamente se crean esas referencias.

Como cualquier otro tipo válido, un método también puede devolver un array.

Ahora ya podemos entender el significado de la lista de argumentos del método `main(String args[])`, es un array de Strings donde cada uno de ellos, y empezando por la posición 0, es un argumento pasado en la línea de órdenes.

```
[belfern@anubis Java]$java HolaJava uno dos tres
```

En este caso `args[0]="uno"`, `args[1]="dos"`, `args[2]="tres"`.

Ejemplo:

```
import java.util.Scanner;
public class cparray {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // atributos datos
        int n;
        int may, men, suma;
        double prom;
        // metodos
        Scanner tecla = new Scanner(System.in);
        int [] ar1 = new int [5]; // declaramos un arreglo de nombre ar1 con
            // cinco elementos
        //Declaramos un for con los siguientes elementos
        // contador i para i = 0 con condicion i <5 , es decir cinco elemen-
        // tos igual que los elementos del arreglo ar1, el contador i se in-
        // crementara de 1 en 1 desde cero hasta que obtiene el valor 4 para
        // ejecutar el bucle ya que con valor cinco lo abandona

        for (int i=0;i<5;i++){
            System.out.print("Ingrese el numero "+(i+1)+" : ");
            n=tecla.nextInt();
            ar1[i]=n;
        }

        // para mostrar el contenido de valores almacenado en el arreglo unidi-
        // mensional, hay que recorrer el arreglo desde su elemento cero hasta su
        // elemento 4, con este for se imprimira el contenido del arreglo ar1 en
        // la misma linea separado por un espacio.
```

```

for(int i=0;i<5;i++){
    System.out.print(ar1[i]+" ");
}
// el siguiente código encuentra el valor mayor en la variable may, el
// valor menor en la variable men y la suma de todos sus elementos en la
// variable suma.
may=ar1[0];
men=ar1[0];
suma = ar1[0];
for (int i=1;i<5;i++){
    if(ar1[i] > may)may=ar1[i];
    if(ar1[i] < men)men=ar1[i];
    suma = suma + ar1[i];
}
// calcula el promedio que es una variable real por esa razón se multiplica suma
por
// 1.0
prom = suma * 1.0 / 5;
// el siguiente código, permite escribir el valor mayor, menor, suma, y
// el promedio
System.out.println("\nmayor: "+may);
System.out.println("menor: "+men);
System.out.println("suma: "+suma);
System.out.println("promedio: "+prom);
// el siguiente código, permite ordenar los valores almacenados en el
// arreglo de menor a mayor con el método de comparación de cada elemen-
// to con todos los elementos del arreglo, para lo cual se emplean dos
// for anidados. Tenga en cuenta que el for más interno varía más rápi-
// do en todos sus elementos que el for externo
int temp;
System.out.println();
for (int i=0;i<4;i++){
    for(int k =(i+1);k<5;k++){
        if(ar1[i]>ar1[k]){
            temp=ar1[i];
            ar1[i]=ar1[k];
            ar1[k]=temp;
        }
    }
}
for(int i=0;i<5;i++){
    System.out.print(ar1[i]+" ");
}
}
}

```



Este programa ejemplo permite cargar cinco elementos a un arreglo en forma desordenada, el programa muestra los elementos ingresados conservando su orden de ingreso, luego calcula el número menor de los elementos ingresados y la suma de sus elementos, los muestra por pantalla, luego calcula el promedio aritmético y lo muestra por pantalla, ordena los cinco datos ingresados de menor a mayor y muestra el arreglo ordenado.

#### 4.6. Cadenas de caracteres.

Una cadena de caracteres es un conjunto de dígitos y letras que se agrupan con el nombre de una variable, en su sentido más explicativo una cadena representa a un arreglo de caracteres alfanuméricos donde su elemento uno ocupa una posición cero en la memoria tan igual que un arreglo, y el elemento dos ocupa la posición uno y así sucesivamente.

En Java existe una clase para representar y manipular cadenas, **la clase String**.

Una vez creado un String no se puede modificar. Se pueden crear instancias de una manera abreviada y sobre ellas se puede utilizar el operador de concatenación +:

```
String frase = "Esta cadena es una frase ";
```

```
String larga =frase + "que se puede convertir en una frase larga.";
```

```
System.out.println(larga);
```

Notas. En Java se ha eliminado la sobrecarga de operadores, el único operador sobrecargado es +.

#### 4.7 Ejercicios

1. Dados 10 números, se pide calcular la suma y el promedio aritmético
2. Dados 10 números, se pide obtener en número menor y mayor
3. Dados 10 números y un divisor, se pide hallar los múltiplos del divisor en cada número ingresado.

4. Ubique un número dado en un arreglo de 10 números ingresados con anterioridad, reporte también su posesión en el arreglo.
5. Ingrese dos matrices de orden 3 x 3 y calcule:
  - Su producto de cada matriz por un número dado
  - la suma de las dos matrices
  - el producto de las dos matrices

## 5. CLASES EN JAVA

La programación orientada a objetos (POO), es una nueva forma de pensar, donde para programar nos acercamos más a lo que acontece en el mundo real, es de allí que abstraemos el diseño de los objetos que se emplearán en la programación donde interactúan los objetos para realizar tareas específicas, esta interacción se realiza mediante el paso de mensajes.

Los objetos encapsulan atributos (datos) que definen su estado y métodos que definen su comportamiento y son instancias de clases o plantillas.

Java nació como un lenguaje de objetos, donde todo son objetos a excepción de los tipos de datos primitivos.

Es importante puntualizar la estructura de este Capítulo, donde se tratan los conceptos relacionados a clase, objeto, tal y como se utilizan en el lenguaje java para construcción de programas, del mismo modo se establece la diferencia en paso por valor y el paso por referencia al utilizar métodos definidos en una clase y por último se muestra cómo se define una clase y cómo se crean objetos que son instancias de alguna de las clases definidas.

### 5.1. Definición de una clase en Java.

En Java una clase se define como un conjunto de objetos que abstraen de la realidad los mismos datos llamados atributos y los mismos métodos que define el comportamiento de la clase, una clase cuando se crea por primera vez hereda de la clase raíz objeto sus datos y sus métodos, entre los cuales podremos citar al método `toString()` muy usados por mostrar datos alfanuméricos a través de la pantalla con `System.out.println()`, el método `wait()`, `notify()` y `notifyAll()` relacionados con los bloqueos cuando se utiliza programación concurrente.

A una clase se representa mediante un rectángulo y tiene la siguiente estructura:

|                  |
|------------------|
| Nombre de Clase  |
| Atributos        |
| <u>Metodos()</u> |

Como se puede observar de la estructura planteada, una clase tiene un nombre que encapsula un conjunto de atributos (Datos) y un conjunto de métodos que definen el comportamiento de la clase.

## 5.2. Atributos.

Los atributos como ya se mencionó son datos especificados en la clase, son conocidos como atributos estáticos, y sus valores en un momento dado define el estado de la clase. Estos valores son inicializados en forma automática por el compilador en función al tipo de dato en el caso de no haber definido valores iniciales en el programa mediante el uso de constructores.

La declaración de un atributo tiene el siguiente formato:

**[modificador de acceso] [static tipoDeDato ] nombre**

El modificador de acceso se emplea para declarar el tipo de visibilidad que tendrá el atributo declarado, esto se consigue con el uso de las siguientes palabras reservadas: `private`, `protected` o `public`

El modificador de acceso **private** se usa para dar visibilidad al atributo solo en la clase que lo define.

El modificador de acceso **protected** se usa para dar visibilidad al atributo en la clase y sus clases herederos.

El modificador de acceso **public** se usa para dar visibilidad al atributo desde cualquier clase public o desde cualquier clase definida en un paquete.

Si no especifica un modificador de clase, por default asume al atributo con un modificador de acceso public.

Se usa la palabra reservada **static** para establecer que el atributo tiene una copia única en todos los objetos de la clase (instancia de clase) por ello a los atributos que se les indica con la palabra reservada **static** se les denomina atributos de clase.

El **tipoDeDato** especifica si el atributo es de tipo primitivo (definidos en el acápite 3.1 de este libro) o de tipo por referencia.

El **nombre** de un atributo debe ser único y el lenguaje java discrimina el uso de minúsculas o mayúsculas por ello si el nombre de un atributo es miAtributo y por error se le llama como MiAtributo estas corresponden a dos nombres de atributo diferentes.

### **Ejemplo 5-1. Uso de atributos y métodos static**

```
public class contInstancias {
    private static int wnumInstancias = 0; //Visualizacion de la clase (Atributo de clase)
    public contInstancias () {
        wnumInstancias++; // Cada vez que ejecuto el constructor se incrementa el
        contador
    }
    public static int cuantasInstancias() {
        return wnumInstancias;
    }
}

public class apliContInstancias {
    public static void main(String[] args) {
        contInstancias objCInstancias = new contInstancias();
        System.out.println("contador =" +objCInstancias.cuantasInstancias());
        contInstancias objCInstancias1 = new contInstancias();
        System.out.println("contador =" +objCInstancias1.cuantasInstancias());
    }
}
```

Otra forma que afecta al formato de la declaración de un atributo es cuando se **declara una constante** (No puede cambiar su valor durante la ejecución del programa) que tiene en su formato la palabra reservada final como por ejemplo cuando se declara el valor de PI.

```
Public static final float PI = 3.1416f;
```

### 5.3. Métodos estáticos o de clase.

Los métodos encapsulados en una clase determinan el comportamiento de la clase y sirven para manipular los valores de los atributos de clase, tiene el siguiente formato:

```
[modificador de acceso] [static] tipoDeDato nombre (parametro)
```

El modificador de acceso se emplea para declarar el tipo de visibilidad que tendrá el método declarado, esto se consigue con el uso de las siguientes palabras reservadas:

```
private, protect o public
```

El modificador de acceso **private** se usa para dar visibilidad al método solo en la clase que lo define.

El modificador de acceso **protect** se usa para dar visibilidad al método en la clase y sus clases herederos.

El modificador de acceso **public** se usa para dar visibilidad al método desde cualquier clase public o desde cualquier clase definida en un paquete.

Si no especifica un modificador de clase, por default asume al método con un modificador de acceso public.

Se usa la palabra reservada **static** para establecer que el método tiene una copia única en todos los objetos de la clase (instancia de clase) por ello a los atributos que se les indica con la palabra reservada static se les denomina métodos de clase. Y no es necesario de instanciarlo para poderlos utilizar.

El **tipoDeDato** especifica si el método es de tipo primitivo (definidos en el acápite 3.1 de este libro) que depende del tipo de datos del atributo que quiere devolver con la palabra reservada return.

Como tipo de datos se considera la palabra reservada void que se utiliza si el método no devuelve algún valor.

Un método que lleva en su estructura la palabra reservada final, indica que no se puede sobrescribir en ninguna clase que se herede de esta clase.

El **nombre** de un método debe ser único y el lenguaje java discrimina el uso de minúsculas o mayúsculas por ello si el nombre de un método es miMetodo y por error se le llama como MiMetodo estas corresponden a dos nombres de método diferentes.

Los **parámetros** que van dentro de los paréntesis se especifican con tipo de dato y nombre separado por comas, y constituyen razón de identificación de los métodos (se les considera como parte del nombre), cuando se declaran varios métodos con el mismo nombre con parámetros diferentes se les conoce como método sobrecargado, depende de la naturaleza del método que se quiere implementar, si no se especifican parámetros queda los paréntesis vacíos.

Los parámetros son considerados como variables locales de los métodos y solo tienen ámbito de significado solo dentro del método.

Los parámetros de los métodos pueden tener el mismo nombre que los atributos de la clase, de ser el caso, para acceder a los atributos de la clase se referencian a partir del objeto actual this.

En ejemplo siguiente observa la utilización de la palabra reservada this, resaltado en negrita.

```
public class Factorial {
    // Atributos
    private int n;
    private int fact;
    // Metodos
    public void EstablecerNumero(int n1)
    {
        setN(n1);
    }
}
```

LIBRO JAVA BÁSICO PARA APRENDICES  
ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hf6e-nj87>

```

public void CalcularFactorial()
{
    setFact(1);
    for (int i = 1; i <= getN(); i++)
    {
        setFact(getFact() * i);
    }
}
public int ObtenerFactorial()
{
    return getFact();
}

/**
 * @return the n
 */
public int getN() {
    return n;
}

/**
 * @param n the n to set
 */
public void setN(int n) {
    this.n = n;
}

/**
 * @return the fact
 */
public int getFact() {
    return fact;
}

/**
 * @param fact the fact to set
 */
public void setFact(int fact) {
    this.fact = fact;
}
}
}

```

El uso de la palabra reservada static en la declaración de un método, hace que este método para implementarlo no requiera de instancia de clase, ya que existe una sola copia



para todas las instancias, lo que hace que para referenciarlo es necesario hacerlo mediante el nombre de la clase que lo contiene, en este caso los métodos declarados de este modo solo podrán usar atributos o métodos declarados como static.

Es importante que lo indicado lo observe en siguiente ejercicio.

### **Ejemplo 5-2. Atributos y Métodos estáticos.**

```
public class contInstancias {
    private static int wnumInstancias = 0; //Visualizacion de la clase (Atributo de clase)
    public contInstancias () {
        wnumInstancias++; // Cada vez que ejecuto el constructor se incrementa el
        contador
    }
    public static int cuantasInstancias() {
        return wnumInstancias;
    }
}

public class apliContInstancias {
    public static void main(String[] args) {
        contInstancias objCInstancias = new contInstancias();
        System.out.println("contador =" +objCInstancias.cuantasInstancias());
        contInstancias objCInstancias1 = new contInstancias();
        System.out.println("contador =" +contInstancias.cuantasInstancias());

        System.out.println("contador =" +contInstancias.cuantasInstancias());
    }
}
```

## **5.4. Constructores**

Un constructor es un método especial que lleva por nombre el mismo nombre que lleva la clase, su importancia es inicializar variables.

Si no se declara un constructor, el lenguaje Java creara uno por defecto, este constructor inicializara las variables (atributos) con valores según su tipo, si es numérico el valor inicial será cero, si es alfanumérico su valor será nulo, etc.

### Ejemplo 5-3. Ejemplo de uso de un constructor

```
public class Ccuenta {
//Atributos
    private String nombre;
    private String cuenta;
    private double saldo;
    private double tipoDeInteres;
// Metodos
    public Ccuenta(){ } // constructor sin parametros
    public Ccuenta(String nom, String cue, double Sal, double tipo){
        asignarNombre(nom);
        asignarCuenta(cue);
        asignarSaldo(Sal);
        asignarTipoDeInteres(tipo);
    }
    public void asignarNombre(String nom){
        if (nom.length() == 0){
            System.out.println("Error: Cadena vacia");
            return;
        }
        nombre = nom;
    }
    public void asignarCuenta(String cue) {
        if (cue.length()== 0){
            System.out.println("Error: Cadena vacia");
            return;
        }
        cuenta=cue;
    }
    public void asignarSaldo(double sal){
        if(sal<0){
            System.out.println("Error: Saldo no valido");
            return;
        }
        saldo = sal;
    }
    public void asignarTipoDeInteres(double tip){
        if(tip<0){
            System.out.println("Error: Tipo no valido");
            return;
        }
        tipoDeInteres = tip;
    }
    public String obtenerNombre(){
```

```

    return nombre;
}
public String obtenerCuenta(){
    return cuenta;
}
public double obtenerSaldo(){
    return saldo;
}
public double estado(){
    return saldo;
}
public double obtenertipoDeInteres(){
    return tipoDeInteres;
}
public void ingreso(double cantidad){
    if (cantidad < 0){
        System.out.println("Error: Cantidad negativa");
        return;
    }
    saldo += cantidad;
}
public void reintegro(double cantidad){
    if (saldo - cantidad < 0){
        System.out.println("Error: no dispone de saldo");
        return;
    }
    saldo -= cantidad;
}
}

```

```

import java.util.Scanner;
public class Test {
    public static void main(String[] args) {
        //Atributos
        String nombre, cuenta;
        double saldo, tipoDeInteres;
        // Metodos
        Scanner teclado = new Scanner(System.in);
        Ccuenta cuenta01 = new Ccuenta();
        Ccuenta cuenta02 = new Ccuenta("un nombre", "Una cuenta", 6000, 3.5);
        cuenta01.asignarNombre("un nombre");
        cuenta01.asignarCuenta("Una cuenta");
        cuenta01.asignarTipoDeInteres(2.5);

        cuenta01.ingreso(12000);
    }
}

```

```

cuenta01.reintegro(3000);

System.out.println(cuenta01.obtenerNombre());
System.out.println(cuenta01.obtenerCuenta());
System.out.println(cuenta01.estado());
System.out.println(cuenta01.obtenertipoDeInteres());
System.out.println();
System.out.println(cuenta02.obtenerNombre());
System.out.println(cuenta02.obtenerCuenta());
System.out.println(cuenta02.estado());
System.out.println(cuenta02.obtenertipoDeInteres());

}

}

```

Observe en el ejercicio propuesto, se han definido dos constructores uno con parámetros y el otro sin parámetros, al constructor sin parámetros se le denomina constructor por defecto.

## 5.5 Creación de objetos

La creación de objetos se hace con el operador new, quien devuelve una referencia al objeto creado. Es esta referencia la que, como ya sabemos, nos sirve para acceder a los atributos y métodos del objeto.

```

public class NewClass {
// Atributos
    char carac;
    int numasc;
    String msg;
// Metodos
    public void estabN(char carac)
    {
        this.carac=carac;
    }
    public void calcula()
    {
        msg="";
        numasc = (int) carac;
    }
}

```

*LIBRO JAVA BÁSICO PARA APRENDICES*  
 ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hf6e-nj87>

```

//(Nota: Dígito numérico (48,57); Mayúscula (65,90); Minúscula (97,122))
if (numasc >= 48 && numasc <= 57)
{
    msg = "digito un numero";
}
else
if (numasc >= 65 && numasc <= 90)
{
    msg="es una mayuscula";
}
else
if (numasc >= 97 && numasc <= 122)
{
    msg="es minuscula";
}
else
{
    msg = "es caracter especial";
}
}

public String mostrarMsg()
{
    return msg+" "+numasc;
}
}

import java.io.*;
public class aplineclass {
public static void main(String[] args) {
    // Atributos
    char a;
    String datoEntrada;
    // Flujo de datos
    InputStreamReader entrada=new InputStreamReader(System.in);
    BufferedReader flujoEntrada=new BufferedReader(entrada);
    NewClass objconv = new NewClass();
    try
    {
        System.out.print("Ingrese caracter: ");
        datoEntrada = flujoEntrada.readLine();
        a=datoEntrada.charAt(0);
        objconv.estabN(a);
        objconv.calcula();
        System.out.println(objconv.mostrarMsg());
    }
}
}

```

```

    }
    catch (IOException error)
    {
        System.err.println("Error " + error.getMessage());
    }
}
}

```

Observe en el ejercicio planteado, la creación del objeto objconv, se hace con el operador new-

### 5.5. Paso por valor y paso por referencia

Cuando se trabaja con métodos que tienen parámetros, estos parámetros se hacen por valor, y como ya se dijo, estos parámetros se comportan como variables locales, por ende, su ámbito se circunscribe al método que lo contiene y esto ocurre si los parámetros del tipo referencia o primitivos.

Si los parámetros son del tipo referencia, la referencia original se copia sobre la referencia del método, esto hace que ambas referencias hagan referencia a la misma instancia de clase (objeto) y las actualizaciones actúen en el objeto.

Si a la referencia dentro del método le asignamos una referencia a otro objeto, al salir del método y desaparecer la referencia dentro del método, las eventuales modificaciones que hayamos hecho sobre el objeto copia desaparecerán.

Para aclarar todo esto observa el siguiente ejemplo y su resultado.

#### **Ejemplo 5-4. Ejemplo de paso de una referencia. Clases:**

##### **Clase uno**

```

public class circulo {

    double areaC;

    public void AreaCirculo(double radio){

```

LIBRO JAVA BÁSICO PARA APRENDICES  
 ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hf6e-nj87>

```
    areaC = 3.1416 *radio*radio;
}
}
```

## Clase 2

```
public class area {
    double varea;

    public void calculaArea(circulo wcir, double wradio){
        wcir = new circulo();
        wcir.AreaCirculo(wradio);
        varea=wcir.areaC;
    }
}
```

## Aplicación (Ejecutable)

```
public class apliArea {
    public static void main(String[] args) {
        area objarea = new area();
        objarea.calculaArea(null, 50.5);
        System.out.println(objarea.varea);
    }
}
```

## 5.6. Sobrecarga de Métodos

Este caso se presenta cuando se definen dos o más métodos o constructores con el mismo nombre, podría presentarse el caso siguiente:

```
Public void calcula (doblé radio) {
```

---

---

```
}
```

```
Public void calcula(){ }
```

Este caso define un método sobrecargado llamado calcula-

## 5.7. Finalización

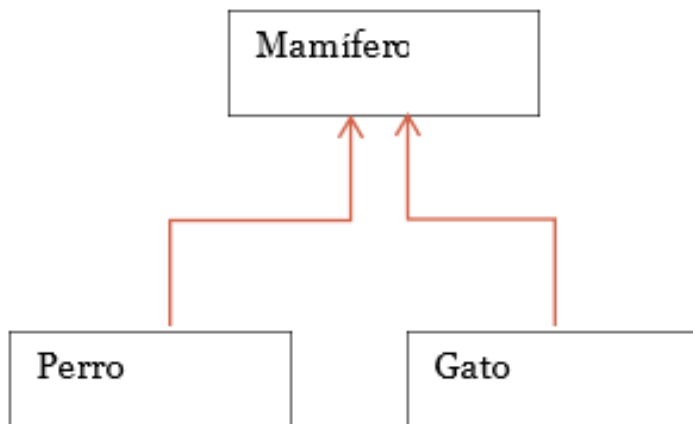
Esta tarea de finalizar completamente a un objeto, se logra llamado a ejecución a su método `finalize()`, este método está definido en la clase `Object`, de cual es heredada por cualquier nueva clase.

Al llamar a ejecución al método `finalize()` del objeto, se comprobara si queda alguna tarea por hacer antes de que el objeto sea destruido, por ejemplo, cerrar los posibles ficheros que tengamos abiertos, cerrar las conexiones de red, etcétera.



## 5.8 Ejercicios

Dado las siguientes estructuras de clases:



### Los atributos de los mamíferos son:

Nombre string (30)

Raza string (20)

Fecha Nacimiento string(10)

peso float

### Los métodos son

Comer()

comunicarse()

Los perros y los gatos tienen los atributos de la clase que a continuación se indican.

### La clase hija perro tiene el atributo específico

Nombre string (30)

Raza string (20)

Fecha Nacimiento string(10) peso float

Lugar de entrenamiento string(30)

### Los métodos son

*LIBRO JAVA BÁSICO PARA APRENDICES*

ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hf6e-nj87>

Comer()

comunicarse()

### **La clase hija gato tiene el atributo específico**

Nombre string (30)

Raza string (20)

Fecha Nacimiento string(10) peso float

altura de salto double

### **Los métodos son**

Comer()

comunicarse()

Con estos datos se pide implementar las clases indicadas definir constructores. Capturar los datos para los perros y los gatos en comunicarse de un perro reportar guau guau y para el gato miau miau

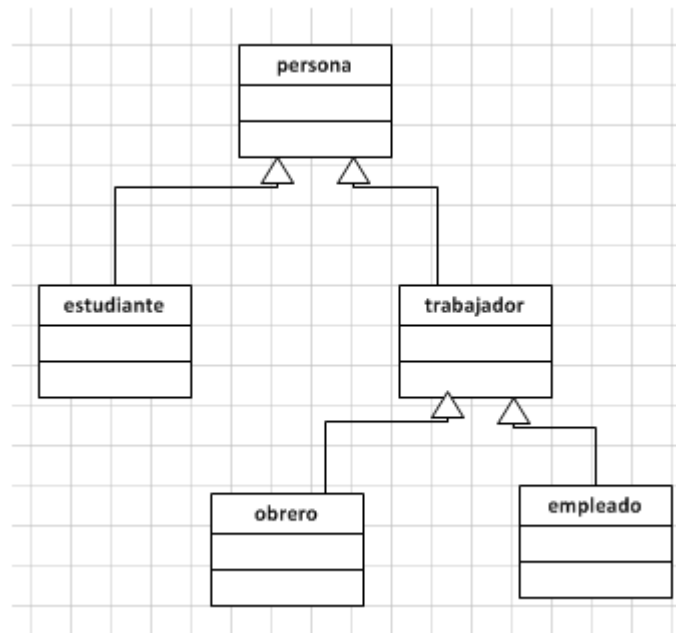
## 6. HERENCIA E INTERFACE EN JAVA

Una de las características más importantes de la programación orientada a objetos es la herencia, que permite que una clase extienda el comportamiento y estado a otra clase, con lo que permite reusar código ya escrito.

Java no soporta herencia múltiple, que es por otro lado fuente de conflictos en los lenguajes que sí la permiten como C++. Sin embargo, en Java se definen lo que se conoce como interface que permiten añadir múltiples características a las clases que los implementan.

Una clase en Java solo puede extender una única clase base, pero puede implementar varios interface.

### 6.1. Herencia



La herencia, llamada también generalización, organiza la estructura en forma arbolescente, como se muestra en la figura, la clase persona se extiende a las clases estudiante

y trabajador, es decir se ha extendido a estas clases. Para que esta ocurra se emplea la palabra reservada **extends**. Cabe recalcar que la clase persona hereda de la clase raíz Object.

La herencia se aplica a los atributos y métodos, es decir la extensión se realiza en esos términos y para referencia al constructor de la clase padre, se emplea la palabra reservada **super**.

En el Ejemplo 6-1 se muestra un ejemplo de herencia.

### **Ejemplo 6-1. Un ejemplo de herencia.**

#### **Clases**

```
public class persona {
// Atributos
    private String dni, nombre, apellidos,nombres, direccion, fono, celular, e_mail;
// Constructor
    public persona( String dni,String apellidos,String nombres, String direccion,String
fono,String celular,String e_mail){
        setDni(dni);
        setApellidos(apellidos);
        setNombres(nombres);
        setDireccion(direccion);
        setFono(fono);
        setCelular(celular);
        setE_Mail(e_mail);
    }
    public void setDni(String dni){
        this.dni = dni;
    }
    public void setApellidos(String apellidos){
        this.apellidos = apellidos;
    }
    public void setNombres(String nombres){
        this.nombres = nombres;
    }
    public void setDireccion(String direccion){
        this.direccion = direccion;
    }
    public void setFono(String fono){
        this.fono = fono;
    }
    public void setCelular(String celular){
        this.celular = celular;
    }
}
```

```

public void setE_Mail(String e_mail){
    this.e_mail = e_mail;
}
public String getdni(){
    return dni;
}
public String getApellidos(){
    return apellidos;
}
public String getNombres(){
    return nombres;
}
public String getDireccion(){
    return direccion;
}
public String getFono(){
    return fono;
}
public String getCelular(){
    return celular;
}
public String getE_Mail(){
    return e_mail;
}
}
public class empleado extends persona {
// atributos
    String fecIngreso;
    double sueBasico;
// Constructor
    public empleado(String dni,String apellidos,String nombres,String direccion, String
fono, String celular,String e_mail,String fIngres,Double sueBasic){
        super(dni,apellidos,nombres,direccion, fono,celular,e_mail);
        setFecIngreso(fIngres);
        setSueBasico(sueBasic);
    }
    public void setFecIngreso(String fIngres){
        fecIngreso = fIngres;
    }
    public void setSueBasico(double sBasico){
        sueBasico = sBasico;
    }
    public String getFecIngreso(){
        return fecIngreso;
    }
}

```

```

    }
    public double getSueBasico(){
        return sueBasico;
    }
}
public class obrero extends persona {
// atributos
    double jornal;
// constructor
    public obrero (String dni,String apel,String nom,String dir, String fono, String celu,
String e_mail, double jornal ){
        super(dni,apel,nom,dir,fono,celu,e_mail);
        setJornal(jornal);
    }
    public void setJornal(double jor){
        jornal = jor;
    }
    public double getJornal(){
        return jornal;
    }
}
}

```

### **Clase aplicación (ejecutable)**

```

import java.util.Scanner;
public class apliPlanilla {
    public static void main(String[] args) {
        // atributos
        String dni, nombre, apellidos,nombres, direccion, fono, celular, e_mail,fecIngres;
        double numHor, sueBru,sueBas,jorn;
        int tipPers;
        // Metodos
        Scanner tecla = new Scanner(System.in);
        System.out.print("ingrese dni: ");
        dni=tecla. botón Next();
        System.out.print("ingrese Apellidos: ");
        apellidos=tecla. botón Next();
        System.out.print("ingrese Nombres: ");
        nombre=tecla. botón Next();
        System.out.print("ingrese direccion: ");
        direccion=tecla. botón Next();
        System.out.print("ingrese fono: ");
        fono=tecla. botón Next();
        System.out.print("ingrese celular: ");
        celular=tecla. botón Next();
        System.out.print("ingrese e_mail: ");
    }
}

```



### 6.1.1. Sobrescritura de variables y métodos.

Si se tiene un atributo en la clase padre, este tiene asociado un nombre y un tipo de datos, en la clase hija se tiene un atributo con el mismo nombre y tipo diferente, en la clase hija muestra una advertencia y cuando se ejecute la hija se dice que el atributo se ha sobre escrito u ocultado. Lo indicado lo podrá ver en el siguiente ejercicio.

```
public class persona {
// Atributos
    private String dni, nombre, apellidos,nombres, direccion, fono, celular, e_mail;
    int edad;
// Constructor
    public persona( String dni,String apellidos,String nombres, String direccion,String
fono,String celular,String e_mail){
        setDni(dni);
        setApellidos(apellidos);
        setNombres(nombres);
        setDireccion(direccion);
        setFono(fono);
        setCelular(celular);
        setE_Mail(e_mail);
    }
    public void setDni(String dni){
        this.dni = dni;
    }
    public void setApellidos(String apellidos){
        this.apellidos = apellidos;
    }
    public void setNombres(String nombres){
        this.nombres = nombres;
    }
    public void setDireccion(String direccion){
        this.direccion = direccion;
    }
    public void setFono(String fono){
        this.fono = fono;
    }
    public void setCelular(String celular){
        this.celular = celular;
    }
    public void setE_Mail(String e_mail){
        this.e_mail = e_mail;
    }
}
```



```

public String getdni(){
    return dni;
}
public String getApellidos(){
    return apellidos;
}
public String getNombres(){
    return nombres;
}
public String getDireccion(){
    return direccion;
}
public String getFono(){
    return fono;
}
public String getCelular(){
    return celular;
}
public String getE_Mail(){
    return e_mail;
}
}
class Entero { public int dato; ...
}
class Real extends Entero { public float dato; ...
}

public class empleado extends persona {
// atributos
    String fecIngreso;
    double sueBasico, edad;
// Constructor
    public empleado(String dni,String apellidos,String nombres,String direccion, String
fono, String celular,String e_mail,String fIngres,Double sueBasic){
        super(dni,apellidos,nombres,direccion, fono,celular,e_mail);
        setFecIngreso(fIngres);
        setSueBasico(sueBasic);
    }
    public void setFecIngreso(String fIngres){
        fecIngreso = fIngres;
    }
    public void setSueBasico(double sBasico){
        sueBasico = sBasico;
    }
}

```

```

public String getFecIngreso(){
    return fecIngreso;
}
public double getSueBasico(){
    return sueBasico;
}
}

public class obrero extends persona {
// atributos
    double jornal;
    double edad;
// constructor
    public obrero (String dni,String apel,String nom,String dir, String fono, String celu,
String e_mail, double jornal ){
        super(dni,apel,nom,dir,fono,celu,e_mail);
        setJornal(jornal);
    }
    public void setJornal(double jor){
        jornal = jor;
    }
    public double getJornal(){
        return jornal;
    }
}
}

```

Una clase hija que extiende a una clase padre puede sobrescribir los métodos de la clase.

### 6.1.2. Sobreescritura de constructores.

Los constructores, no son métodos normales sino especiales y invocan al momento de instanciar una clase y también se pueden sobre escribir en momento de la extensión padre y cuando se ejecute el código que se ejecutara es la hija y no del padre.

### 6.1.3. Vinculación dinámica.

Cuando la instanciación de una clase y ejecución de los demás métodos se hace en tiempo de ejecución dependiente al valor de una variable asignada anteriormente, a este caso se le denomina vinculación dinámica, para mayor comprensión observe el siguiente ejemplo:

## Clases Padre e hijas

```
public class Circulo {
    double radio, area;
    public void calculaArea(double wradio){
        area = 3.1416 * wradio * wradio;
    }
    public double mostrararea(){
        return area;
    }
}

public class cuadrado {
    double lado, area;
    public void calculaArea(double wlado){
        area = wlado * wlado;
    }
    public double mostrarArea(){
        return area;
    }
}

import java.util.Scanner;
public class apliArea {
    public static void main(String[] args) {
        // atributos
        int wtipo;
        double wradio, wlado;
        Scanner teclado = new Scanner(System.in);
        System.out.println("ingrese 1 si es circulo");
        System.out.println("ingrese 2 si es cuadrado");
        wtipo = teclado.nextInt();
        if (wtipo == 1){
            System.out.print("ingrese radio: ");
            wradio = teclado.nextDouble();
            Circulo objarea = new Circulo();
            objarea.calculaArea(wradio);
            System.out.println (objarea.mostrararea());
        }
        else {
            System.out.print("ingrese lado: ");
            wlado = teclado.nextDouble();
            cuadrado objarea = new cuadrado();
            objarea.calculaArea(wlado);
            System.out.println (objarea.mostrarArea());
        }
    }
}
```

```
}  
  
}  
  
}
```

## Aplicación (Ejecuciónj)

```
/*  
 * To change this template, choose Tools | Templates  
 * and open the template in the editor.  
 */  
  
/**  
 *  
 * @author Manuel  
 */  
import java.util.*;  
public class apliHerencia {  
  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
        String dni, nombre, apellidos,nombres, direccion, fono, celular, e_mail, fecIngres;  
        double numHor, sueBru,sueBas,jorn;  
        int tipPers;  
        // Metodos  
        Scanner tecla = new Scanner(System.in);  
        System.out.print("ingrese dni: ");  
        dni=tecla.next();  
        System.out.print("ingrese Apellidos: ");  
        apellidos=tecla.next();  
        System.out.print("ingrese Nombres: ");  
        nombre=tecla.next();  
        System.out.print("ingrese direccion: ");  
        direccion=tecla.next();  
        System.out.print("ingrese fono: ");  
        fono=tecla.next();  
        System.out.print("ingrese celular: ");  
        celular=tecla.next();  
        System.out.print("ingrese e_mail: ");  
        e_mail=tecla.next();  
        System.out.print("ingrese horas trajadas: ");  
        numHor=tecla.nextDouble();
```

```

System.out.print("ingrese 1...Empledos, 2.....para Obreros: ");
tipPers=tecla.nextInt();
if (tipPers == 1){
    System.out.print("ingrese fecha Ingreso: ");
    fecIngres=tecla.next();
    System.out.print("ingrese sueldo basico: ");
    sueBas=tecla.nextDouble();
    empleado objemp = new
empleado(dni,apellidos,nombre,direccion,fono,celular,e_mail,fecIngres,sueBas);
    System.out.println("dni: "+objemp.getDni());
    System.out.println("Nombres: "+objemp.getNombres());
    System.out.println("apellidos: "+objemp.getApellidos());
    System.out.println("Fecha Ingreso: "+objemp.getFecIngreso());
    System.out.println("total Bruto: "+(objemp.getSueBasico()*numHor)/240.0);

}
if (tipPers == 2){
    System.out.print("ingrese Jornal: ");
    jorn=tecla.nextDouble();
    obrero objobre = new
obrero(dni,apellidos,nombre,direccion,fono,celular,e_mail,jorn);
    System.out.println("dni: "+objobre.getDni());
    System.out.println("Nombres: "+objobre.getNombres());
    System.out.println("apellidos: "+objobre.getApellidos());
    System.out.println("total Bruto: "+objobre.getJornal()*numHor);
}
}
}
}

```

#### 6.1.4. El operador instanceof.

El operador instanceof sirve para hacer la prueba de compatibilidad de la naturaleza entre una instancia de clase y una clase, deben ser de tipos compatibles, en caso contrario habrá un error en tiempo de compilación este error se puede mostrar en la siguiente instrucción:

**System.out.println(objcuadrado instanceof circulo);**

La que dara un error por tipos incompatibles en tiempo de compilación.

Si no existe error imprimirá el mensaje de TRUE.

```

public class circulo {
double radio, area;
public void calculaArea(double wradio){
    area = 3.1416 * wradio * wradio;
}
public double mostrararea(){
    return area;
}
}

public class cuadrado {
double lado, area;
public void calculaArea(double wlado){
    area = wlado * wlado;
}
public double mostrarArea(){
    return area;
}
}

public class apliinstanceof {
    public static void main(String[] args) {
        cuadrado objcuadrado = new cuadrado();
        circulo objcirculo = new circulo();
        System.out.println(objcuadrado instanceof cuadrado);
        System.out.println(objcirculo instanceof circulo);
    }
}

```

Este programa produce la siguiente salida:

```

true
true

```

### 6.1.1. Clases abstractas.

Cuando existen métodos abstractas en una clase y el método no se implementa en dicha clase, entonces se trata una clase abstracta.

Una clase abstracta, puede contener un método abstracto que no se implementa en dicha clase, pero cuando se declara un método como abstracto la clase que lo contiene debe ser obligatoriamente abstracto. Una clase abstracta no se puede instanciar y las clases que la

extiendan deben sobrescribir los métodos abstractos para ello emplearán al comienzo del método abstracto de la clase hija la cláusula @override. Vea el código del ejemplo siguiente:

```
public abstract class SeleccionFutbol {
//Atributos
    protected int id;
    protected String nombre;
    protected String apellidos;
    protected int edad;
// Constructor
    public SeleccionFutbol (int id, String nombre, String apellidos, int edad){
        setId(id);
        setNombre(nombre);
        setApellidos(apellidos);
        setEdad(edad);
    }
// Metodos
    public void setId(int id){
        this.id = id;
    }
    public void setNombre(String nombre){
        this.nombre = nombre;
    }
    public void setApellidos(String apellidos){
        this.apellidos = apellidos;
    }
    public void setEdad(int edad){
        this.edad = edad;
    }
    public void viajar() {
        System.out.println("Viajar (Clase Padre)");
    }

    public void concentrarse() {
        System.out.println("Concentrarse (Clase Padre)");
    }
}
```

**// IMPORTANTE -> METODO ABSTRACTO => no se implementa en la clase abstracta pero si en las clases hijas**

```
public abstract void entrenamiento();
```

```
public void partidoFutbol() {
    System.out.println("Asiste al Partido de Fútbol (Clase Padre)");
}
```

```
public String getNombre(){
```

LIBRO JAVA BÁSICO PARA APRENDICES  
ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hf6e-nj87>

```

        return nombre;
    }
    public String getApellidos(){
        return apellidos;
    }
}

public class entrenador extends SeleccionFutbol {
// Atributos
    private int idFederacion;
// Constructor
    public entrenador(int id, String nomb, String apell, int edad, int idFederacion){
        super (id,nomb,apell,edad);
        setIdFederacion(idFederacion);
    }
// Metodos
    public void setIdFederacion(int idFederacion){
        this.idFederacion=idFederacion;
    }
    @Override
    public void entrenamiento() {
        System.out.println("Dirige un entrenamiento (Clase Entrenador)");
    }

    @Override
    public void partidoFutbol() {
        System.out.println("Dirige un Partido (Clase Entrenador)");
    }

    public void planificarEntrenamiento() {
        System.out.println("Planificar un Entrenamiento");
    }
}

public class jugadores extends SeleccionFutbol {
// Atributos
    private int dorsal;
    private String demarcacion;
// Constructor
    public jugadores (int id, String nomb, String apell, int edad, int dorsal, String
demarcacion){
        super (id,nomb,apell,edad);
        setDorsal(dorsal);
        setDemarcacion(demarcacion);
    }
}

```



```

// Metodos
public void setDorsal(int dorsal){
    this.dorsal = dorsal;
}
public void setDemarcacion(String demarcacion){
    this.demarcacion = demarcacion;
}
@Override
public void entrenamiento() {
    System.out.println("Realiza un entrenamiento (Clase Futbolista)");
}
}

public class masajistas extends SeleccionFutbol {
    // Atributos
    private String titulacion;
    private int aniosExperiencia;
    // Constructor
    public masajistas (int id, String nomb, String apell, int edad, String titula, int
aniosExper){
        super (id, nomb, apell,edad);
        setTitulacion(titula);
        setAnioExperiencia(aniosExper);
    }
    // Metodos
    public void setTitulacion(String titulacion){
        this.titulacion = titulacion;
    }
    public void setAnioExperiencia(int aniosExperiencia){
        this.aniosExperiencia = aniosExperiencia;
    }
    @Override
    public void entrenamiento() {
        System.out.println("Da asistencia en el entrenamiento (Clase Masajista)");
    }
}

public class testSeleccionFutbol {
    public static void main(String[] args) {
        SeleccionFutbol sf = new entrenador (1, "Ricardo", "Gareca", 50, 28489);
        System.out.print(sf.getNombre()+ " "+sf.getApellidos()+" ");
        sf.entrenamiento();
        sf = new jugadores (2, "Paolo", "Guerrero", 26, 9, "Centro delantero");
        System.out.print(sf.getNombre()+ " "+sf.getApellidos()+" ");
        sf.entrenamiento();
    }
}

```

```

        sf = new masajistas(3, "Raúl", "Martinez", 41, "Licenciado en Fisioterapia", 18);
        System.out.print(sf.getNombre()+" "+sf.getApellidos()+" ");
        sf.entrenamiento();
    }
}

```

## 6.2. Interfaces

Las interfaces nos dice qué debemos cumplir para que al construir una clase se pueda calificar con un nombre determinado. Por ejemplo, la interface del programa que se muestra nos dice:

```

public interface IAnimal {
    int valor=5;

    /**
     * Método Comunicarse, sera implementado por todas las clases concretas que
     hereden de la clase Animal
     */

    public void comunicarse();
}

```

Dado que una interface no es una clase, no la puede instanciar y los métodos declarados en una instancia no se pueden implementar, sirven para tratarlas como un método abstracto en todas las clase hijas que la heredan, se sobreescriben con la clausula **@override**.

Las interfaces se declaran con la palabra reservada **implements** en la clase padre conforme se muestra a continuación.

Definamos esta interface:

```

public abstract class Animal implements IAnimal {
    private String nombre;

    /**
     * Constructor de la clase Animal
     * @param nombre
     */
    public Animal (String nombre){
        this.nombre=nombre;
        System.out.println("Constructor Animal, " + "nombre del animal : "+this.nombre);
    }
}

```

LIBRO JAVA BÁSICO PARA APRENDICES  
 ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hf6e-nj87>

```

}

/**
 * Retorna el valor de nombre
 * @return
 */
public String getNombre(){
return nombre;
}

/**
 * Metodo Abstracto tipoAnimal, la implementación depende
 * de las clases concretas que extiendan la clase Animal
 */
public abstract void tipoAnimal();
}

```

### **INTERFACE: IAnimal**

```

public interface IAnimal {
int valor=5;

```

```

/**
 * Método Comunicarse, sera implementado por todas las clases concretas que hereden
de la clase Animal
 */
public void comunicarse();
}

```

### **public class Perro extends Animal {**

```

/**
 * @param nombre
 */
public Perro(String nombre) {
super(nombre);
System.out.println("Constructor perro, nombre : "+nombre);
}
@Override
public void tipoAnimal() {
System.out.println("Tipo Animal : Es un Perro");
}

public void comunicarse(){
System.out.println("Metodo comunicarse : El perro Ladra... Guau Guau");
}
}

```

```
}
```

```
public class Gato extends Animal {
```

```
/**
```

```
 * Constructor explicito clase Gato
```

```
 * @param nombre
```

```
 */
```

```
 public Gato(String nombre) {
```

```
 super(nombre);//envia el parametro a el constructor de la clase padre
```

```
 System.out.println("Constructor Gato, nombre : "+nombre);
```

```
 }
```

```
@Override
```

```
 public void tipoAnimal() {
```

```
 System.out.println("Tipo Animal : Es un Gato");
```

```
 }
```

```
 public void comunicarse(){
```

```
 System.out.println("Metodo comunicarse : El gato maulla... Miau Miau");
```

```
 }
```

```
}
```

```
public class TestAnimal {
```

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

```
 /**Creamos anim, un objeto Perro de tipo Animal*/
```

```
 Animal anim= new Perro("goliath") ;
```

```
 anim.tipoAnimal();
```

```
 anim.comunicarse();
```

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

```
 /**Creamos perro, un objeto Perro de tipo Perro*/
```

```
 Perro perro=new Perro("hercules");
```

```
 perro.tipoAnimal();
```

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

```
 /**Creamos animalPolimorfico, un objeto perro de tipo Animal
```

```
 * asignamos una referencia ya existente*/
```

```
 Animal animalPolimorfico=perro;
```

```
 animalPolimorfico.tipoAnimal();
```

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

```
 /**reassignamos la referencia del objeto anim a el objeto perro
```

```
 * esto es valido ya que ambos son de tipo Perro*/
```

```
 perro=(Perro) anim;
```

```
 perro.tipoAnimal();
```

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

*LIBRO JAVA BÁSICO PARA APRENDICES*

ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hf6e-nj87>

```

/**Creamos gat, un objeto Gato de tipo Animal*/
Animal gat=new Gato("pochi");
gat.tipoAnimal();
gat.comunicarse();
System.out.println();

/**Creamos cat, un objeto Gato de tipo IAnimal
 * Para esto aplicamos polimorfismo usando la Interface*/
IAnimal cat = new Gato("pitufa");
cat.comunicarse();

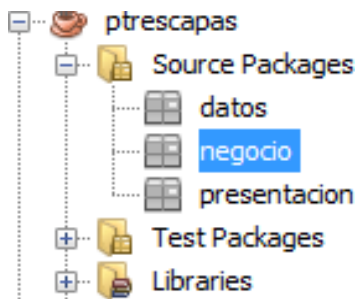
System.out.println("\nConstante en la interfaz Animal : "+IAnimal.valor);
}
}

```

Finalmente, una interface puede extender a otro u otros interfaces. Y si una clase extiende este interface, debe implementar todos los métodos definidos en todas las interfaces.

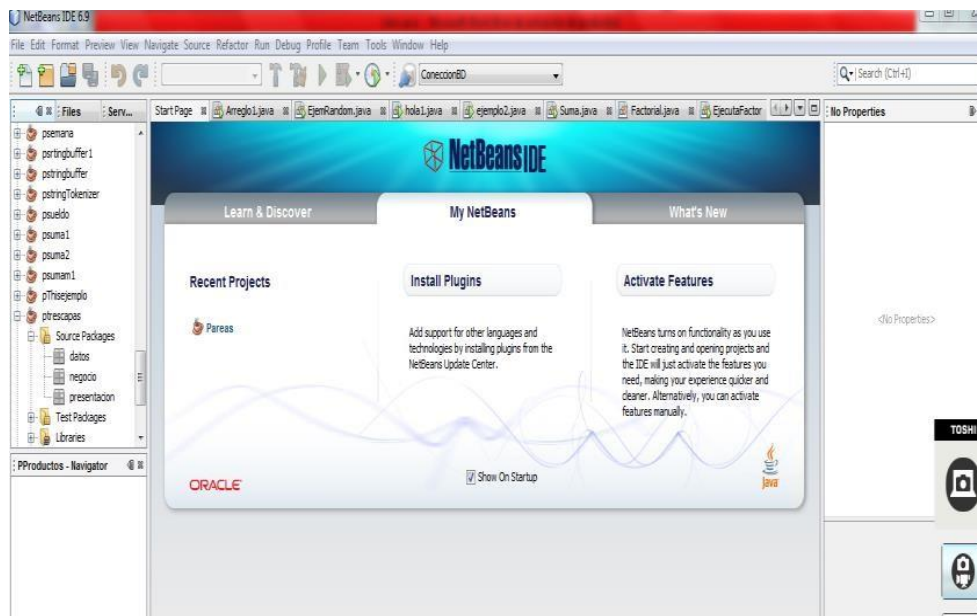
### 6.3. Paquetes.

Los paquetes usados en el lenguaje de programación java, tienen como finalidad proveer al diseñador de sistemas herramientas que permitan dar una organización al diseño de sus proyectos, logrando de este modo mayor claridad en la estructura de la programación. Así por ejemplo se facilita la implementación del desarrollo de proyectos informáticos aplicando la metodología de las tres capas y que son las capas de presentación, capa de datos, y la capa del negocio. Quedando la organización en el entorno java como la siguiente:

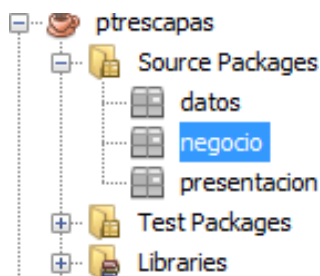


Para crear un paquete en el entorno java de neatbeans, existen dos métodos, el primer método consiste en lo siguiente:

Entrar al entorno neatbeans de java, esta acción colocará en su pantalla la siguiente imagen:

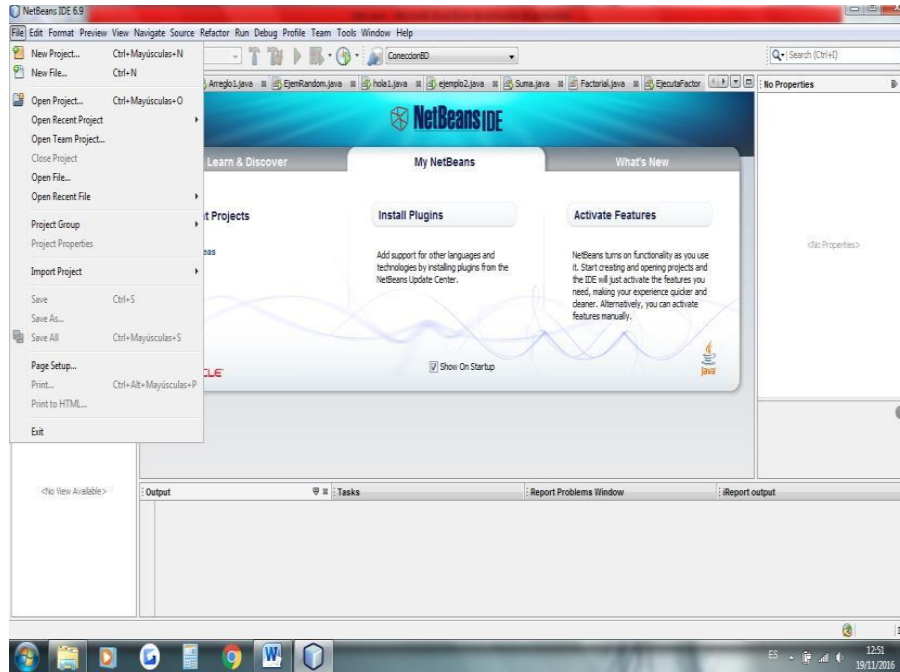


Estando en este entorno, creamos la aplicación java la que colocara en su pantalla la siguiente imagen:

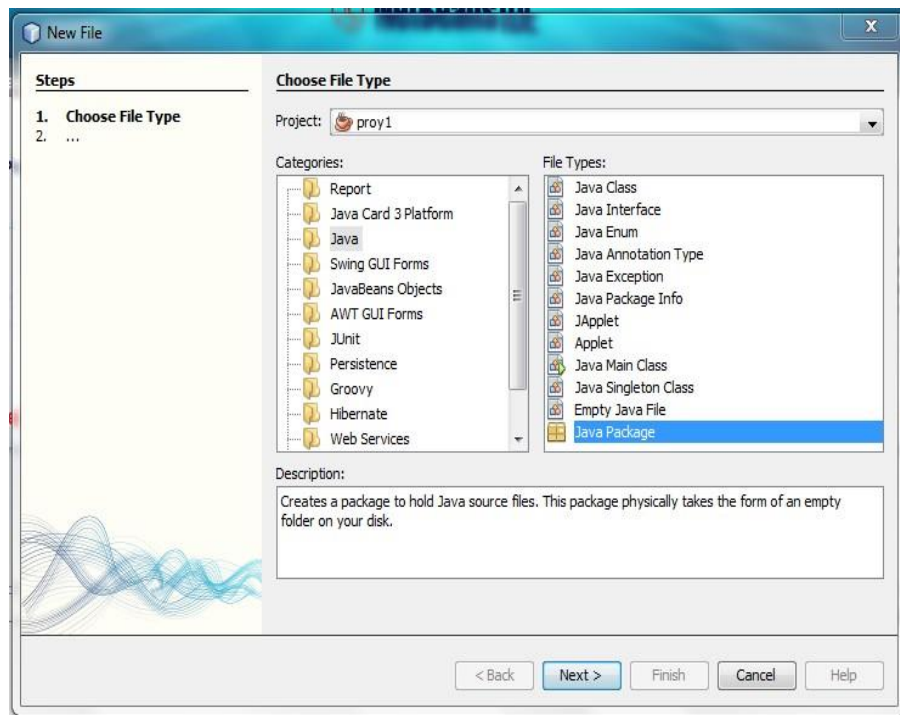


El proyecto creado tiene por nombre **ptrescapas** y luego en esta imagen dar click en **Source Packages**, resaltándose este texto:

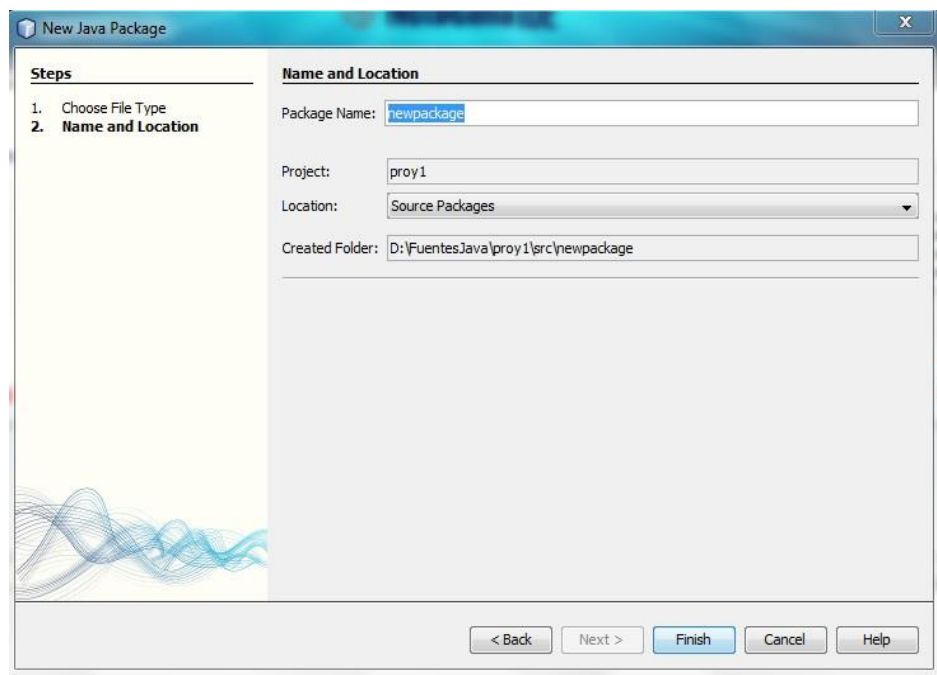
El primer método para crear un paquete, consiste en dar click a la opción File del entorno neatbeans, esto coloca en su pantalla la siguiente imagen:



En este menú que se despliega, hacer click en la opción New File, y aparecerá en su pantalla la imagen siguiente:



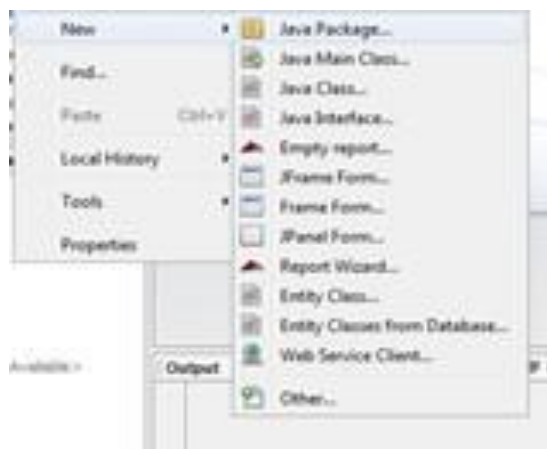
Donde seleccionara la opción **java**, y luego en las opciones de la derecha la opción **Java Package** esta acción colocara en su pantalla la siguiente imagen:



Es en esta imagen que se le dará el nombre del paquete.

El método 2, consiste en lo siguiente:

Creado el proyecto con nombre **ptrescapas** y luego en esta imagen dar clic en **Source Packages**, que resalta este texto hacer clic en el **botón derecho** del mouse y luego hacer clic en la opción **New**, esta acción colocara en su pantalla la siguiente imagen:





Cada clase contendrá clases que guarden relación con la información del paquete, las cuales tendrán acceso entre ellas, para acceder a la información de clases de otros paquetes se debe importar con la sentencia **import**.

El uso de la sentencia import, se explicará a continuación:

### Capa de datos

```
package datos;

/**
 *
 * @author Manuel Jesus Abanto Morales
 */
import java.sql.*;
public class DatosBD {
// Atributos
String mensaje="";
ResultSet result;
String wcod, wnom, wdir, wfono, wcorr;
boolean wencontrado;
String url = "jdbc:odbc:ODBC_facturas";
String usuario = "MANUEL-PC";
String password = "";
Statement stmt = null;
Statement stmt1 = null;
Statement stmt2 = null;
public void conectarBD(){
//Carga del driver
try {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
}
catch(java.lang.ClassNotFoundException ex) {
System.err.print("Problemas al cargar el driver");
System.err.println(ex.getMessage());
}
try {
//Creando la conexion a la BD
Connection conexion = DriverManager.getConnection(url, usuario, password);
//Lanzando consultas
stmt = conexion.createStatement();
result = stmt.executeQuery("SELECT * FROM clientes");
mensaje= "Coneccion ok";
System.out.println(mensaje);
}
}
}
```

```

    }
    catch(SQLException exc) {
    System.err.println(exc.getMessage());
    }
    }

public void Modificar(String cod, String wnom1, String wdir1, String wfono1, String
wcorr1){

    try{
        wencontrado = true;
        Connection conexion2 = DriverManager.getConnection(url, usuario, password);
        stmt2 = conexion2.createStatement();
        if(wencontrado){
            stmt2.execute("UPDATE Clientes set nomcli =
"+wnom1+"",dircli="+wdir1+",telcli="+wfono1+",corrcli="+wcorr1+"where codcli
="+cod+"");
            mensaje = "Modificado";
            System.out.println(mensaje);
        }
        else {
            mensaje="Cliente no ingresado";
            System.out.println(mensaje);
        }
    }catch (SQLException exc){
        System.err.println(exc.getMessage());
    }
    }

public void Grabar(String cod, String wnom1, String wdir1, String wfono1, String
wcorr1){

    try{
        wencontrado = false;
        Connection conexion1 = DriverManager.getConnection(url, usuario, password);
        stmt1 = conexion1.createStatement();
        if(!wencontrado){
            stmt1.execute("INSERT INTO Clientes values
("+cod+", "+wnom1+", "+wdir1+", "+wfono1+", "+wcorr1+"");
            mensaje = "Nuevo";
            System.out.println(mensaje);
        }
        else {
            mensaje="Cliente ya ingresado";
            System.out.println(mensaje);
        }
    }
}

```

```

    }catch (SQLException exc){
        System.err.println(exc.getMessage());
    }
}
public void buscaCliente(String cod){
    try{
        wencontrado= false;

        while(result.next()){
            wcod = result.getString("codcli");
            if (cod.equals(wcod)){ wencontrado = true;
            wnom = result.getString("nomcli");
            wdir = result.getString("dircli");
            wfono = result.getString("telcli");
            wcorr = result.getString("corrcli");
// System.out.println(wcod + "\t" + wnom + "\t" + wdir+ "\t" + wtip + "\t" + wmail);

        }
        System.out.println(wcod + "\t" + wnom);
        }
    }catch (SQLException exc){
        System.err.println(exc.getMessage());
    }
}
public String mostrarNom(){
    return wnom;
}
public String mostrarDir(){
    return wdir;
}
public String mostrarFono(){
    return wfono;
}
public String mostrarCorr(){
    return wcorr;
}
public boolean mostraEncontrado(){
    return wencontrado;
}
}
}

/*

```

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

package datos;

/\*\*

\*

\* @author Manuel Abanto Morales

\*/

import java.sql.\*;

public class DatosBDproductos {

// Atributos

String mensaje="";

ResultSet result;

String wcod, wnom, wume, wpre, wstock;

boolean wencontrado;

String url = "jdbc:odbc:ODBC\_facturas";

String usuario = "MANUEL-PC";

String password = "";

Statement stmt = null;

Statement stmt1 = null;

Statement stmt2 = null;

public void conectarBD(){

//Carga del driver

try {

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

}

catch(java.lang.ClassNotFoundException ex) {

System.err.print("Problemas al cargar el driver");

System.err.println(ex.getMessage());

}

try {

//Creando la conexion a la BD

Connection conexion = DriverManager.getConnection(url, usuario, password);

//Lanzando consultas

stmt = conexion.createStatement();

result = stmt.executeQuery("SELECT \* FROM productos");

mensaje= "Coneccion ok";

LIBRO JAVA BÁSICO PARA APRENDICES

ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hf6e-nj87>

```

    System.out.println(mensaje);
}
catch(SQLException exc) {
    System.err.println(exc.getMessage());
}
}

public void grabar(String cod, String wnom1, String wume1, String wpre1, String
wstock1){

    try{
        wencontrado= false;

        while(result.next()){
            wcod = result.getString("codpro");
            if (cod.equals(wcod)){ wencontrado = true;
            wnom = result.getString("nompro");
            wume = result.getString("umepro");
            wpre = result.getString("prepro");
            wstock = result.getString("stopro");
            }
// System.out.println(wcod + "\t" + wnom + "\t" + wdir+ "\t" + wtip + "\t" + wmail);
    System.out.println(wcod + "\t" + wnom);
        }
        Connection conexion1 = DriverManager.getConnection(url, usuario, password);
        stmt1 = conexion1.createStatement();
        if(!wencontrado){
            stmt1.execute("INSERT INTO productos values
("+cod+", ""+wnom1+"", ""+wume1+"", ""+wpre1+"", ""+wstock1+"")");
            mensaje = "Nuevo";
            System.out.println(mensaje);
        }
        else {
            mensaje="producto ya ingresado";
            System.out.println(mensaje);
        }
    }catch (SQLException exc){
        System.err.println(exc.getMessage());
    }
}

public void Modificar(String cod, String wnom1, String wume1, String wpre1, String
wstock1){

    try{
        wencontrado= false;

```

```

while(result.next()){
    wcod = result.getString("codpro");
    if (cod.equals(wcod)){ wencontrado = true;
    wnom = result.getString("nompro");
    wume = result.getString("umepro");
    wpre = result.getString("prepro");
    wstock = result.getString("stopro");
    }
// System.out.println(wcod + "\t" + wnom + "\t" + wdir+ "\t" + wtip + "\t" + wmail);
    System.out.println(wcod + "\t" + wnom);
}
Connection conexion2 = DriverManager.getConnection(url, usuario, password);
stmt2 = conexion2.createStatement();
if(wencontrado){
    stmt2.execute("UPDATE productos set
nompro='"+wnom1+"',umepro='"+wume1+"',prepro='"+wpre1+"',stopro='"+wstock1+""");
    mensaje = "Modificado";
    System.out.println(mensaje);
}
else {
    mensaje="producto no ingresado";
    System.out.println(mensaje);
}
}catch (SQLException exc){
    System.err.println(exc.getMessage());
}
}
}
public void buscaProducto(String cod){
    try{
        wencontrado= false;

        while(result.next()){
            wcod = result.getString("codpro");
            if (cod.equals(wcod)){
                wencontrado = true;
                wnom = result.getString("nompro");
                wume = result.getString("umepro");
                wpre = result.getString("prepro");
                wstock = result.getString("stopro");
                System.out.println(wcod + "\t" + wnom + "\t" + wume+ "\t" +wpre + "\t" +wstock);

            }
            System.out.println(wcod + "\t" + wnom);
        }
    }catch (SQLException exc){

```

```

        System.err.println(exc.getMessage());
    }
}
public String mostrarNom(){
    return wnom;
}
public String mostrarUme(){
    return wume;
}
public String mostrarPre(){
    return wpre;
}
public String mostrarStock(){
    return wstock;
}
public boolean mostraEncontrado(){
    return wencontrado;
}
}
}

```

En el paquete datos se implementa la clase DatosBD como se muestra en el segmento de código mostrado.

### **En la capa aplicación**

Se muestra el siguiente diseño de pantalla que es operado por el código que se muestra a continuación:

**Factura:**

Cliente:   Fecha:

Dirección:

Producto:  Cantidad

| Código | Nombre | Unid.Med. | Precio | Cantidad | Subtotal |
|--------|--------|-----------|--------|----------|----------|
|        |        |           |        |          |          |
|        |        |           |        |          |          |
|        |        |           |        |          |          |

Tot.Bruto

IGV.

Tot.Netto

```

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

```

```

/*
 * frmFactura.java
 *
 * Created on 24-jun-2016, 17:04:10
 */

```

```

package aplicacion;

```

```

/**
 *
 * @author Manuel
 */
import datos.*;
import java.util.Vector;
import javax.swing.table.DefaultTableModel;
public class frmFactura extends javax.swing.JFrame {
    int swContador = 0;
    DefaultTableModel tablam = new DefaultTableModel();
    double totbru=0, igv, total;
    /** Creates new form frmFactura */
    public frmFactura() {
        initComponents();
    }
}

```

LIBRO JAVA BÁSICO PARA APRENDICES  
 ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hf6e-nj87>



```

/** This method is called from within the constructor to
* initialize the form.
* WARNING: Do NOT modify this code. The content of this method is
* always regenerated by the Form Editor.
*/
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    txtTBru = new javax.swing.JTextField();
    jLabel4 = new javax.swing.JLabel();
    txtcant = new javax.swing.JTextField();
    jLabel9 = new javax.swing.JLabel();
    jLabel3 = new javax.swing.JLabel();
    txtCPro = new javax.swing.JTextField();
    txtfec = new javax.swing.JTextField();
    jLabel8 = new javax.swing.JLabel();
    txtNcli = new javax.swing.JTextField();
    txtdir = new javax.swing.JTextField();
    txtCCli = new javax.swing.JTextField();
    jLabel2 = new javax.swing.JLabel();
    txtNFac = new javax.swing.JTextField();
    jLabel1 = new javax.swing.JLabel();
    jScrollPane1 = new javax.swing.JScrollPane();
    tabprod = new javax.swing.JTable();
    txtigv = new javax.swing.JTextField();
    jLabel6 = new javax.swing.JLabel();
    txttot = new javax.swing.JTextField();
    jLabel7 = new javax.swing.JLabel();
    btnAgregar = new javax.swing.JButton();
    jLabel5 = new javax.swing.JLabel();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

    txtTBru.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            txtTBruActionPerformed(evt);
        }
    });

    jLabel4.setText("Tot.Bruto");

    jLabel9.setText("Cantidad ");

```

```

jLabel3.setText("Cliente:");

jLabel8.setText("Producto:");

txtCCli.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        txtCCliActionPerformed(evt);
    }
});

jLabel2.setFont(new java.awt.Font("Tahoma", 0, 14));
jLabel2.setText("Fecha:");

jLabel1.setFont(new java.awt.Font("Tahoma", 1, 24));
jLabel1.setText("Factura:");

tabprod.setModel(new javax.swing.table.DefaultTableModel(
    new Object [][] {
        {null, null, null, null, null, null},
        {null, null, null, null, null, null},
        {null, null, null, null, null, null}
    },
    new String [] {
        "Codigo", "Nombre", "Unid.Med.", "Precio", "Cantidad", "Subtotal"
    }
) {
    Class[] types = new Class [] {
        java.lang.String.class, java.lang.String.class, java.lang.String.class,
        java.lang.String.class, java.lang.String.class, java.lang.String.class
    };

    public Class getColumnClass(int columnIndex) {
        return types [columnIndex];
    }
});
jScrollPane1.setViewportViewView(tabprod);

jLabel6.setText("Tot.Neto");

jLabel7.setText("Direccion:");

btnAgregar.setText("Agregar");
btnAgregar.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnAgregarActionPerformed(evt);
    }
});

```

```

    }
  });

  jLabel5.setText("IGV.");

  javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
  getContentPane().setLayout(layout);
  layout.setHorizontalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(layout.createSequentialGroup()
        .addGap(159, 159, 159)
        .addComponent(jLabel1)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(txtNFac, javax.swing.GroupLayout.PREFERRED_SIZE, 70,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(189, Short.MAX_VALUE))
      .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
        .addContainerGap()
        .addComponent(jLabel4)
        .addComponent(jLabel5)
        .addComponent(jLabel6)
        .addGap(30, 30, 30)

      .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
false)
        .addComponent(txtp)
        .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()
          .addGap(10, 10, 10)
          .addComponent(txtigv)
          .addComponent(txtTBru, javax.swing.GroupLayout.DEFAULT_SIZE, 103,
Short.MAX_VALUE)))
      .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
        .addContainerGap()
        .addComponent(jLabel7)

      .addGroup(javax.swing.GroupLayout.Alignment.RELATED)

```

```

        .addComponent(txtDir, javax.swing.GroupLayout.DEFAULT_SIZE, 334,
Short.MAX_VALUE))
        .addGroup(layout.createSequentialGroup()
        .addComponent(jLabel3)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(txtCCli, javax.swing.GroupLayout.PREFERRED_SIZE,
66, javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(txtNcli, javax.swing.GroupLayout.DEFAULT_SIZE, 151,
Short.MAX_VALUE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(jLabel2)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(txtfec, javax.swing.GroupLayout.PREFERRED_SIZE, 72,
javax.swing.GroupLayout.PREFERRED_SIZE)))
        .addGap(126, 126, 126))
        .addGroup(layout.createSequentialGroup()
        .addContainerGap()
        .addComponent(jLabel8)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(txtCPro, javax.swing.GroupLayout.PREFERRED_SIZE, 75,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(18, 18, 18)
        .addComponent(jLabel9)
        .addGap(18, 18, 18)
        .addComponent(txtcant, javax.swing.GroupLayout.PREFERRED_SIZE, 93,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(41, 41, 41)
        .addComponent(btnAgrega)
        .addContainerGap(98, Short.MAX_VALUE))
        .addComponent(jScrollPane1, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, 521, Short.MAX_VALUE)
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
        .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
        .addComponent(jLabel1)

```

```

        .addComponent(txtNFac, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGap(13, 13, 13)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
        .addComponent(jLabel2)
        .addComponent(txtfec, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(jLabel3)
        .addComponent(txtCcli, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(txtNcli, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
        .addComponent(jLabel7)
        .addComponent(txtmdir, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGap(28, 28, 28)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jLabel8)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
        .addComponent(txtCPro, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(jLabel9)
        .addComponent(txtcant, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(btnAgregar)))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
31, Short.MAX_VALUE)
        .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE,
118, javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

```

```

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
    .addComponent(txtTBru, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
    .addComponent(jLabel4))
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
    .addComponent(txtigv, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
    .addComponent(jLabel5))
.addGap(9, 9, 9)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
    .addComponent(txttot, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
    .addComponent(jLabel6))
);

pack();
} // </editor-fold>

private void txtTBruActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

private void txtCCLIActionPerformed(java.awt.event.ActionEvent evt) {
    // atributos
    String wcod=txtCCLI.getText();
    DatosBD objconecion = new DatosBD();
    objconecion.conectarBD();
    objconecion.buscaCliente(wcod);
    if(objconecion.mostraEncontrado()){
        txtNcli.setText(objconecion.mostrarNom());
        txtdir.setText(objconecion.mostrarDir());
    }
}

private void btnAgregarActionPerformed(java.awt.event.ActionEvent evt) {
    DatosBDproductos objpro = new DatosBDproductos();
    Vector datos = new Vector();
    swContador++;
}

```

```

if (swContador == 1){
    tablam.addColumn("Codigo");
    tablam.addColumn("Nombre");
    tablam.addColumn("Uni.Med.");
    tablam.addColumn("Precio");
    tablam.addColumn("Cantidad");
    tablam.addColumn("Total");
    tabprod.setModel(tablam);
}
objpro.conectarBD();
objpro.buscaProducto(txtCPro.getText());
if (objpro.mostraEncontrado()){
    System.out.println("Carga datos al vector");
    datos.addElement(txtCPro.getText());
    datos.addElement(objpro.mostrarNom());
    datos.addElement(objpro.mostrarUme());
    datos.addElement(objpro.mostrarPre());
    datos.addElement(txtcant.getText());
    double a, b, c;
    a=Double.parseDouble(objpro.mostrarPre());
    b=Double.parseDouble(txtcant.getText());
    c=a * b;
    datos.addElement(String.valueOf(c));
    tablam.addRow(datos);
    tabprod.setModel(tablam);
    totbru = totbru +c;
    igv = totbru * 0.18;
    total = totbru + igv;
    txtTBru.setText(String.valueOf(totbru));
    txtigv.setText(String.valueOf(igv));
    txttot.setText(String.valueOf(total));
    int longi = datos.size();
    for (int i = (longi - 1); i>= 0; i--) {
        System.out.println (datos.elementAt (i) );
    }
}
}

/**
 * @param args the command line arguments
 */
// TODO add your handling code here:
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new frmFactura().setVisible(true);
        }
    });
}

```

```

    }
    });
}

// Variables declaration - do not modify
private javax.swing.JButton btnAgregar;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JLabel jLabel8;
private javax.swing.JLabel jLabel9;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JTable tabprod;
private javax.swing.JTextField txtCCli;
private javax.swing.JTextField txtCPro;
private javax.swing.JTextField txtNFac;
private javax.swing.JTextField txtNcli;
private javax.swing.JTextField txtTBru;
private javax.swing.JTextField txtcant;
private javax.swing.JTextField txtdir;
private javax.swing.JTextField txtfec;
private javax.swing.JTextField txtigv;
private javax.swing.JTextField txttot;
// End of variables declaration

}

```

Si del paquete aplicación se quiere acceder a la clase DatosBD, en código de la clase frmFactura se incluirá la sentencia import como se muestra resaltado de amarillo.

### Capa Clientes

```

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/*
 * frmFactura.java
 *
 * Created on 24-jun-2016, 17:04:10

```

LIBRO JAVA BÁSICO PARA APRENDICES  
 ISBN: 978-958-53018-8-7 DOI: <https://doi.org/10.34893/hj6e-nj87>



```

*/

package aplicacion;

/**
 *
 * @author Manuel
 */
import datos.*;
import java.util.Vector;
import javax.swing.table.DefaultTableModel;
public class frmFactura extends javax.swing.JFrame {
    int swContador = 0;
    DefaultTableModel tablam = new DefaultTableModel();
    double totbru=0, igv, total;
    /** Creates new form frmFactura */
    public frmFactura() {
        initComponents();
    }

    /** This method is called from within the constructor to
    * initialize the form.
    * WARNING: Do NOT modify this code. The content of this method is
    * always regenerated by the Form Editor.
    */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        txtTBru = new javax.swing.JTextField();
        jLabel4 = new javax.swing.JLabel();
        txtcant = new javax.swing.JTextField();
        jLabel9 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        txtCPro = new javax.swing.JTextField();
        txtfec = new javax.swing.JTextField();
        jLabel8 = new javax.swing.JLabel();
        txtNcli = new javax.swing.JTextField();
        txtdir = new javax.swing.JTextField();
        txtCCli = new javax.swing.JTextField();
        jLabel2 = new javax.swing.JLabel();
        txtNFac = new javax.swing.JTextField();
        jLabel1 = new javax.swing.JLabel();
        jScrollPane1 = new javax.swing.JScrollPane();
        tabprod = new javax.swing.JTable();
    }
}

```

```

txtigv = new javax.swing.JTextField();
jLabel6 = new javax.swing.JLabel();
txttot = new javax.swing.JTextField();
jLabel7 = new javax.swing.JLabel();
btnAgrega = new javax.swing.JButton();
jLabel5 = new javax.swing.JLabel();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

txtTBru.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        txtTBruActionPerformed(evt);
    }
});

jLabel4.setText("Tot.Bruto");

jLabel9.setText("Cantidad ");

jLabel3.setText("Cliente:");

jLabel8.setText("Producto:");

txtCCli.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        txtCCliActionPerformed(evt);
    }
});

jLabel2.setFont(new java.awt.Font("Tahoma", 0, 14));
jLabel2.setText("Fecha:");

jLabel1.setFont(new java.awt.Font("Tahoma", 1, 24));
jLabel1.setText("Factura:");

tabprod.setModel(new javax.swing.table.DefaultTableModel(
    new Object [][] {
        {null, null, null, null, null, null},
        {null, null, null, null, null, null},
        {null, null, null, null, null, null}
    },
    new String [] {
        "Codigo", "Nombre", "Unid.Med.", "Precio", "Cantidad", "Subtotal"
    }
) {

```

```

    Class[] types = new Class [] {
        java.lang.String.class, java.lang.String.class, java.lang.String.class,
        java.lang.String.class, java.lang.String.class, java.lang.String.class
    };

    public Class getColumnClass(int columnIndex) {
        return types [columnIndex];
    }
});
jScrollPane1.setViewportView(tabprod);

jLabel6.setText("Tot.Neto");

jLabel7.setText("Direccion:");

btnAgregar.setText("Agregar");
btnAgregar.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnAgregarActionPerformed(evt);
    }
});

jLabel5.setText("IGV.");

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addGap(159, 159, 159)
            .addComponent(jLabel1)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(txtNFac, javax.swing.GroupLayout.PREFERRED_SIZE, 70,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(189, Short.MAX_VALUE))
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
            .addGap(242, 242, 242)

            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(jLabel4)
                .addComponent(jLabel5)
                .addComponent(jLabel6))
            .addGap(30, 30, 30)

```

```

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING,
false)
    .addComponent(txttot)
    .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()
    .addGap(10, 10, 10)
    .addComponent(txtigv))
    .addComponent(txtTBru, javax.swing.GroupLayout.DEFAULT_SIZE, 103,
Short.MAX_VALUE)))
    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
    .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
    .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()
    .addComponent(jLabel7)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
    .addComponent(txtmdir, javax.swing.GroupLayout.DEFAULT_SIZE, 334,
Short.MAX_VALUE))
    .addGroup(layout.createSequentialGroup()
    .addComponent(jLabel3)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
    .addComponent(txtCcli, javax.swing.GroupLayout.PREFERRED_SIZE,
66, javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
    .addComponent(txtNcli, javax.swing.GroupLayout.DEFAULT_SIZE, 151,
Short.MAX_VALUE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
    .addComponent(jLabel2)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
    .addComponent(txtfec, javax.swing.GroupLayout.PREFERRED_SIZE, 72,
javax.swing.GroupLayout.PREFERRED_SIZE)))
    .addGap(126, 126, 126))
    .addGroup(layout.createSequentialGroup()
    .addContainerGap()
    .addComponent(jLabel8)
    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

```

```

        .addComponent(txtCPro, javax.swing.GroupLayout.PREFERRED_SIZE, 75,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(18, 18, 18)
        .addComponent(jLabel9)
        .addGap(18, 18, 18)
        .addComponent(txtcant, javax.swing.GroupLayout.PREFERRED_SIZE, 93,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(41, 41, 41)
        .addComponent(btnAgregar)
        .addContainerGap(98, Short.MAX_VALUE))
        .addComponent(jScrollPane1, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, 521, Short.MAX_VALUE)
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel1)
            .addComponent(txtNFac, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(13, 13, 13)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel2)
            .addComponent(txtfec, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(jLabel3)
            .addComponent(txtCcli, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(txtNcli, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel7)
            .addComponent(txtdir, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(28, 28, 28)

```

```

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addComponent(jLabel8)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
    .addComponent(txtCPro, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
    .addComponent(jLabel9)
    .addComponent(txtcant, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
    .addComponent(btnAgregar)))
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
31, Short.MAX_VALUE)
.addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE,
118, javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
    .addComponent(txtTBru, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
    .addComponent(jLabel4))
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
    .addComponent(txtigv, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
    .addComponent(jLabel5))
.addGap(9, 9, 9)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
    .addComponent(txttot, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
    .addComponent(jLabel6))
);

pack();
} // </editor-fold>

private void txtTBruActionPerformed(java.awt.event.ActionEvent evt) {

```

```

        // TODO add your handling code here:
    }

private void txtCCLIActionPerformed(java.awt.event.ActionEvent evt) {
    // atributos
    String wcod=txtCCli.getText();
    DatosBD objconeccion = new DatosBD();
    objconeccion.conectarBD();
    objconeccion.buscaCliente(wcod);
    if(objconeccion.mostraEncontrado()){
        txtNcli.setText(objconeccion.mostrarNom());
        txtdir.setText(objconeccion.mostrarDir());
    }
}

private void btnAgregarActionPerformed(java.awt.event.ActionEvent evt) {
    DatosBDproductos objpro = new DatosBDproductos();
    Vector datos = new Vector();
    swContador++;
    if (swContador == 1){
        tablam.addColumn("Codigo");
        tablam.addColumn("Nombre");
        tablam.addColumn("Uni.Med.");
        tablam.addColumn("Precio");
        tablam.addColumn("Cantidad");
        tablam.addColumn("Total");
        tabprod.setModel(tablam);
    }
    objpro.conectarBD();
    objpro.buscaProducto(txtCPro.getText());
    if (objpro.mostraEncontrado()){
        System.out.println("Carga datos al vector");
        datos.addElement(txtCPro.getText());
        datos.addElement(objpro.mostrarNom());
        datos.addElement(objpro.mostrarUme());
        datos.addElement(objpro.mostrarPre());
        datos.addElement(txtcant.getText());
        double a, b, c;
        a=Double.parseDouble(objpro.mostrarPre());
        b=Double.parseDouble(txtcant.getText());
        c=a * b;
        datos.addElement(String.valueOf(c));
        tablam.addRow(datos);
        tabprod.setModel(tablam);
        totbru = totbru +c;
    }
}

```

```

    igv = totbru * 0.18;
    total = totbru + igv;
    txtTBru.setText(String.valueOf(totbru));
    txtigv.setText(String.valueOf(igv));
    txttot.setText(String.valueOf(total));
    int longi = datos.size();
    for (int i = (longi - 1); i >= 0; i--) {
        System.out.println (datos.elementAt (i) );
    }
}
}

/**
 * @param args the command line arguments
 */ // TODO add your handling code here:
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new frmFactura().setVisible(true);
        }
    });
}

```

```

// Variables declaration - do not modify
private javax.swing.JButton btnAgrega;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JLabel jLabel8;
private javax.swing.JLabel jLabel9;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JTable tabprod;
private javax.swing.JTextField txtCCLI;
private javax.swing.JTextField txtCPro;
private javax.swing.JTextField txtNFac;
private javax.swing.JTextField txtNcli;
private javax.swing.JTextField txtTBru;
private javax.swing.JTextField txtcant;
private javax.swing.JTextField txtdir;
private javax.swing.JTextField txtfec;
private javax.swing.JTextField txtigv;

```



```

private javax.swing.JTextField txttot;
// End of variables declaration

}

/*
* To change this template, choose Tools | Templates
* and open the template in the editor.
*/

/*
* BDProductos.java
*
* Created on 24-jun-2016, 18:04:47
*/

package clientes;

/**
*
* @author Manuel
*/
import datos.*;
public class BDProductos extends javax.swing.JFrame {

    /** Creates new form BDProductos */
    public BDProductos() {
        initComponents();
    }

    /** This method is called from within the constructor to
    * initialize the form.
    * WARNING: Do NOT modify this code. The content of this method is
    * always regenerated by the Form Editor.
    */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

```

```

jLabel1 = new javax.swing.JLabel();
jLabel2 = new javax.swing.JLabel();
jLabel3 = new javax.swing.JLabel();
jLabel4 = new javax.swing.JLabel();
jLabel5 = new javax.swing.JLabel();
jLabel6 = new javax.swing.JLabel();
txtcod = new javax.swing.JTextField();
txtnom = new javax.swing.JTextField();
txtume = new javax.swing.JTextField();
txtpre = new javax.swing.JTextField();
txtsto = new javax.swing.JTextField();
btnNuevo = new javax.swing.JButton();
btnGrabar = new javax.swing.JButton();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

jLabel1.setFont(new java.awt.Font("Tahoma", 1, 24)); // NOI18N
jLabel1.setText("Mantenimiento tabla de productos");

jLabel2.setFont(new java.awt.Font("Tahoma", 0, 14)); // NOI18N
jLabel2.setText("Codigo:");

jLabel3.setFont(new java.awt.Font("Tahoma", 0, 14)); // NOI18N
jLabel3.setText("Nombre:");

jLabel4.setFont(new java.awt.Font("Tahoma", 0, 14)); // NOI18N
jLabel4.setText("Unid. Med.:");

jLabel5.setFont(new java.awt.Font("Tahoma", 0, 14)); // NOI18N
jLabel5.setText("Precio:");

jLabel6.setFont(new java.awt.Font("Tahoma", 0, 14)); // NOI18N
jLabel6.setText("Stock:");

btnNuevo.setText("Nuevo");
btnNuevo.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnNuevoActionPerformed(evt);
    }
});

btnGrabar.setText("Grabar");
btnGrabar.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnGrabarActionPerformed(evt);
    }
});

```



```

        .addComponent(txtpre,
javax.swing.GroupLayout.DEFAULT_SIZE, 92, Short.MAX_VALUE)
        .addComponent(txttume,
javax.swing.GroupLayout.PREFERRED_SIZE, 74,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(txtsto)))
    .addGroup(layout.createSequentialGroup()
        .addComponent(btnNuevo)
        .addGap(18, 18, 18)
        .addComponent(btnGrabar))))
    .addContainerGap(105, Short.MAX_VALUE))
);
layout.setVerticalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(layout.createSequentialGroup()
        .addContainerGap()
        .addComponent(jLabel1)
        .addGap(18, 18, 18)

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel2)
            .addComponent(txtcod, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)

            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jLabel3)
                .addComponent(txtnom, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(18, 18, 18)

                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addComponent(jLabel4)
                    .addComponent(txttume, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGap(18, 18, 18)

                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

```

```

        .addComponent(jLabel5)
        .addComponent(txtpre, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASSEL
INE)
        .addComponent(jLabel6)
        .addComponent(txtsto, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGap(49, 49, 49)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASSEL
INE)
        .addComponent(btnNuevo)
        .addComponent(btnGrabar))
        .addContainerGap(93, Short.MAX_VALUE))
    );

    pack();
} // </editor-fold>

```

```

private void btnNuevoActionPerformed(java.awt.event.ActionEvent evt) {
    txtcod.setText(null);
    txtnom.setText(null);
    txtume.setText(null);
    txtpre.setText(null);
    txtsto.setText(null);
    txtcod.requestFocus();
}

```

```

private void btnGrabarActionPerformed(java.awt.event.ActionEvent evt) {
    String wcod = txtcod.getText();
    String wnom1 = txtnom.getText();
    String wume1 = txtume.getText();
    String wpre1 = txtpre.getText();
    String wstock1 = txtsto.getText();
    DatosBDproductos objconectar = new DatosBDproductos();
    objconectar.conectarBD();
    objconectar.grabar(wcod, wnom1, wume1, wpre1, wstock1);
}

```

```
/**
```

```

* @param args the command line arguments
*/
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new BDProductos().setVisible(true);
        }
    });
}

```

```

// Variables declaration - do not modify
private javax.swing.JButton btnGrabar;
private javax.swing.JButton btnNuevo;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JTextField txtcod;
private javax.swing.JTextField txtnom;
private javax.swing.JTextField txtpre;
private javax.swing.JTextField txtsto;
private javax.swing.JTextField txtume;
// End of variables declaration

```

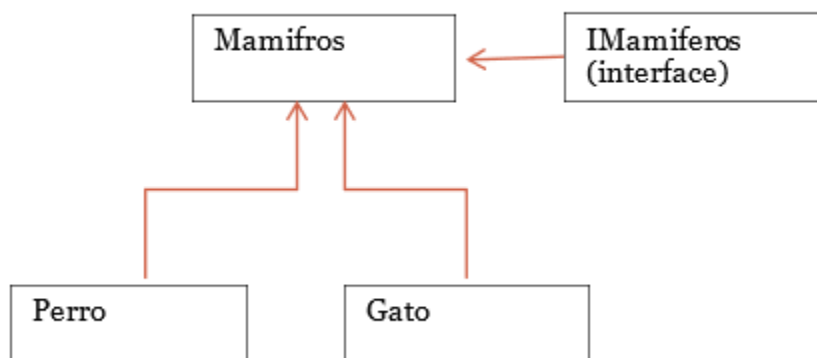
```

}

```

## 6.4 Ejercicios

1.- Dado la siguiente estructura de clases:



### **Los atributos de los mamíferos son:**

Nombre string (30)  
Raza string (20)  
TipoAnimal string(20)  
Fecha Nacimiento string(10)  
peso float

### **Los métodos son**

Comer()  
TipoAnimal()  
comunicarse()  
Los perros y los gatos heredan los atributos de la clase

### **Definición de la interface IMamiferos**

#### **Definición de la interface IMamiferos**

/\*\*

**\* Método Comunicarse, sera implementado por todas las clases concretas que hereden de la clase Mamiferos**

\*/

#### **Metodos:**

comunicarse()

### **La clase hija perro tiene el atributos específico**

Lugar de entrenamiento string(30)

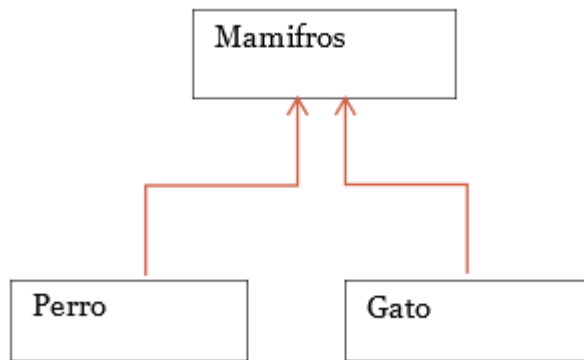
### **La clase hija gato tiene el atributo específico**

altura de salto double

### **Se pide:**

Con estos datos se pide implemetar las clases indicadas definir constructores Capturar los datos para los peros y los gatos en comunicarse de un perro reportar guau guau y para el gato miau miau.

2.- Dado la siguiente estructura de clases:



**Los atributos de los mamíferos son:**

- Nombre string (30)
- Raza string (20)
- TipoAnimal string(20)
- Fecha Nacimiento string(10)
- peso float

**Los métodos son**

- Comer()
- TipoAnimal()
- comunicarse()

Nota: La clase padre mamíferos es una clase abstracta

Los perros y los gatos heredan los atributos de la clase padre

**La clase hija perro tiene el atributos específico**

Lugar de entrenamiento string(30)

**La clase hija gato tiene el atributo específico**

altura de salto double

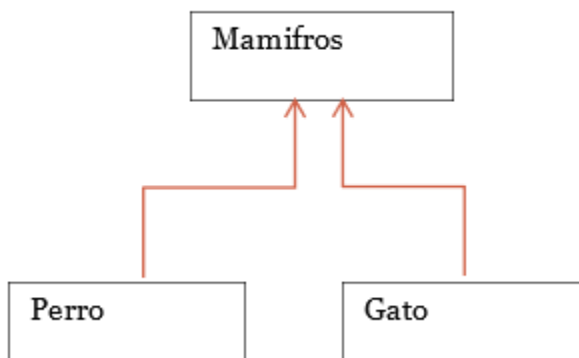
El método de las clases hijas perro y gato son abstractas

**Se pide:**



Con estos datos se pide implemetar las clases indicadas definir constructores Capturar los datos para los peros y los gatos en comunicarse de un perro reportar guau guau y para el gato miau miau

### 3.- Dado la siguiente estructura de clases:



#### Los atributos de los mamíferos son:

Nombre string (30)  
Raza string (20)  
TipoAnimal string(20)  
Fecha Nacimiento string(10)  
peso float

#### Los métodos son

Comer()  
TipoAnimal()  
comunicarse()

Nota: La clase padre mamíferos es una clase abstracta

Los perros y los gatos heredan los atributos de la clase padre

#### La clase hija perro tiene el atributo específico

Lugar de entrenamiento string(30)

#### La clase hija gato tiene el atributo específico

altura de salto double

El método de las clases hijas perro y gato son abstractas

**Se pide:**

Con estos datos se pide implementar las clases indicadas definir constructores Capturar los datos para los perros y los gatos, en comunicarse de un perro reportar guau guau y para el gato miau miau

4.- Diseñar un sistema de pedidos para una tienda que atiende por pedidos para lo cual debe considerar los siguientes requerimientos:

- Diseñar su base de datos considerar en forma mínima las tablas Cliente, productos, pedidos cada una con sus atributos más representativos
- Implementar su base de datos en SQL
- Implementar el sistema de pedidos con Java, utilizando la metodología de las tres capas, el sistema debe contemplar la captura de datos y el registro de la información en cada tabla del sistema.

## 7. EXCEPCIONES EN JAVA

El manejo de las excepciones java es presentado en este capítulo.

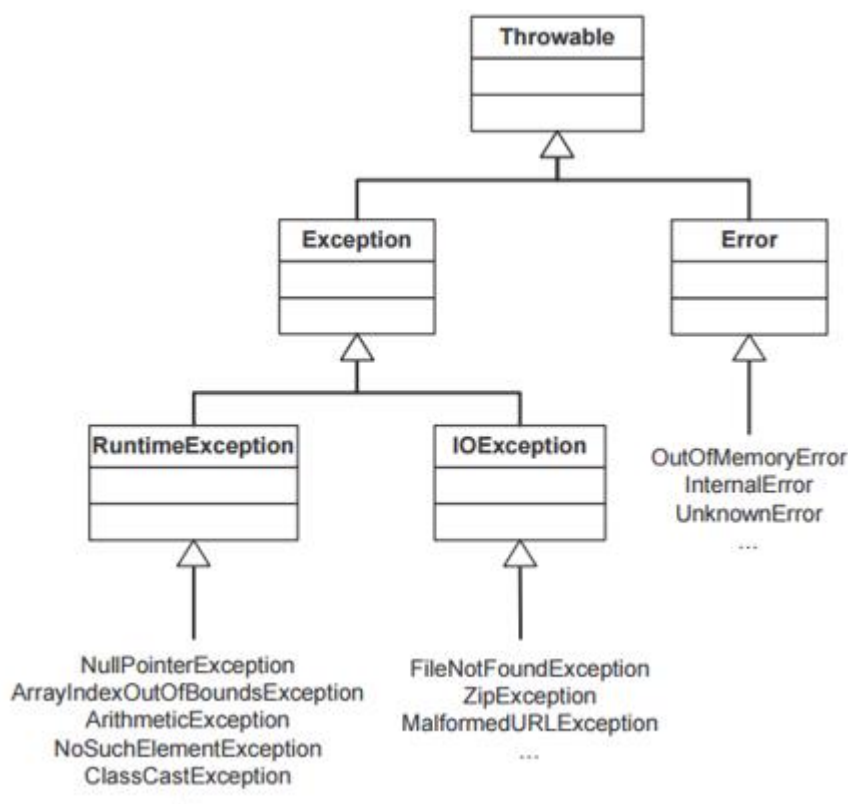
### 7.1. Qué es una excepción.

En Java y en cualquier lenguaje de programación existen errores en tiempo de ejecución (cuando se esta ejecutando el programa) a estos errores se denominan excepciones, y esto ocurre cuando se produce un error en alguna de las instrucciones de nuestro programa, como por ejemplo cuando se hace una división entre cero, cuando un objeto es 'null' y no puede serlo, cuando se trata de abrir un archivo que no existe.

En Java los errores se dan un tratamiento cuando se producen mediante instrucciones creadas para tal fin.

### 7.2. Tipos de excepciones.

Existe una jeraquia de clases para lanzar una excepción en java:



Donde se puede observar que existen dos tipos de excepciones, la **Exception** y el **error**, en la **exception**, existen dos grupos como son el **RuntimeException** producida por los programadores como un punto nulo de excepción, por el índice de un array mal definido, por una expresión aritmética, por ningún elemento exceptuado, o por una excepción de clase y el **IOException** producida no por los programadores como archivo no encontrado, por una excepción de cierre, una URL mal formada.

### 7.3. Gestión de excepción: try...catch...finally.

Toda excepción será tratada con la sentencia `try{...} catch{...}`, si se lanza una excepción, el control de ejecución del programa actualiza la pila del programa y el control se trasladará al código `catch {Exception e.....}` para atrapar la excepción generada.

#### Ejemplo 7-1. Ejemplo de tratamiento de excepciones.

```
import java.sql.*;
public class ConeccionBD extends javax.swing.JFrame {

    /** Creates new form ConeccionBD */
    public ConeccionBD() {
        initComponents();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jLabel1 = new javax.swing.JLabel();
        btnConectar = new javax.swing.JButton();
        jLabel2 = new javax.swing.JLabel();
        txtMsg = new javax.swing.JTextField();
        btnIrMenu = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
```

```

jLabel1.setFont(new java.awt.Font("Tahoma", 1, 14));
jLabel1.setText("Coneccion a la BD");

btnConectar.setText("Conectar");
btnConectar.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnConectarActionPerformed(evt);
    }
});

btnIrMenu.setText("Home");
btnIrMenu.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnIrMenuActionPerformed(evt);
    }
});

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addGap(114, 114, 114)
            .addComponent(jLabel1)
            .addGap(92, 92, 92)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED_SIZE,
180, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGroup(layout.createSequentialGroup()
                    .addComponent(txtMsg, javax.swing.GroupLayout.PREFERRED_SIZE,
228, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGap(99, 99, 99))
            ));
layout.setVerticalGroup(

```

```

        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addGap(108, 108, 108)
            .addComponent(jLabel1)
            .addGap(27, 27, 27)

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(btnConectar)
            .addComponent(btnIrMenu)
            .addGap(42, 42, 42)
            .addComponent(txtMsg, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jLabel2)
            .addContainerGap(57, Short.MAX_VALUE))
        );

        pack();
    } // </editor-fold>

```

```

    private void btnConectarActionPerformed(java.awt.event.ActionEvent evt) {
        String url = "jdbc:odbc:ODBC_Ventas";
        String usuario = "MANUEL-PC";
        String password = "";
        Statement stmt = null;
        //Carga del driver
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        }
        catch(java.lang.ClassNotFoundException ex) {
            System.err.print("Problemas al cargar el driver");
            System.err.println(ex.getMessage());
        }
        try {
            //Creando la conexion a la BD
            Connection conexion = DriverManager.getConnection(url, usuario, password);
            //Lanzando consultas
            stmt = conexion.createStatement();
            ResultSet result = stmt.executeQuery("SELECT * FROM Clientes");
            txtMsg.setText("Coneccion ok");

            //while(result. botón Next()){
            //    id = result.getInt("EmployeeID");
            //    nombrel = result.getString("LastName");

```

```

// nombref = result.getString("FirstName");
// System.out.println(id + "\t" + nombrel + "\t" + nombref);
//}

}
catch(SQLException exc) {
System.err.println(exc.getMessage());
}

}

private void btnIrMenuActionPerformed(java.awt.event.ActionEvent evt) {
    PMenu Menu = new PMenu();
    Menu.setVisible(true);
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new ConeccionBD().setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private javax.swing.JButton btnConectar;
private javax.swing.JButton btnIrMenu;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JTextField txtMsg;
// End of variables declaration

}

```

Este código establece una conexión por ODBC a una base de datos SQL, y funciona muy bien, aquí se presenta en negrita la aplicación del comando try---catch, estudiala con cuidado.

## BIBLIOGRAFÍA

- Belmonte Fernández, O. (2 de 12 de 2005). *introducción al lenguaje de programación Java*. Recuperado el 30 de 10 de 2016, de [www3.uji.es/~belfern/pdidoc/IX26/Documentos/introJava.pdf](http://www3.uji.es/~belfern/pdidoc/IX26/Documentos/introJava.pdf)
- M.C. M. Nakayama C, A. Y. (s.f.). *Entorno y lenguaje de programación - Sitio Odin.fi-b.unam.mx*. Recuperado el 31 de 10 de 2016, de [http://odin.fi-b.unam.mx/salac/practicaspoo/P01\\_POO\\_EntornoLenguaje.pdf](http://odin.fi-b.unam.mx/salac/practicaspoo/P01_POO_EntornoLenguaje.pdf)
- Rodríguez, A. (s.f.). *Netbeans, Eclipse, JCreator, JBuilder... ¿Cuál es el mejor entorno de desarrollo (IDE) para Java? (CU00613B)*. Recuperado el 30 de 10 de 2016, de [http://www.aprenderaprogramar.com/index.php?option=com\\_content&view=article&id=398:netbeans-eclipse-jcreator-jbuilder-icual-es-el-mejor-entorno-de-desarrollo-ide-para-java-cu00613b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188](http://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=398:netbeans-eclipse-jcreator-jbuilder-icual-es-el-mejor-entorno-de-desarrollo-ide-para-java-cu00613b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188)
- Salinas, S. (10 de 05 de 2015). *Java (Programación Orientada a Objetos): Java (Lenguaje de ...* Recuperado el 31 de 10 de 2016, de <http://zoledadsalinas.blogspot.pe/2015/05/que-es-java.html>
- TRIPOD. (s.f.). *Java desde Cero - Intranet DCC*. Recuperado el 30 de 10 de 2016, de <http://moisesrbb.tripod.com/java1.htm>