CHOUCAIR
Business Centric Testing

# Workshop de Automatización de Pruebas

## NIVELACIÓN EN JAVA

Parte 2

Noviembre 2025

Fundamentos de Java para Automatización

CHOUCAIR
Business Centric Testing

# Vectores y matrices

## Vectores

Los vectores son una estructura de datos que permite almacenar un grupo de datos de un mismo tipo. También son conocidos popularmente como arrays.

*Tomado de la página de campusmvp.es*

También es habitual llamar **matrices** a los vectores que trabajan con dos dimensiones.

```
//Forma 1:
int vectorNumeros1[] = new int[10];
//Forma 2:
int []vectorNumeros2 = new int[10];
//Forma 3:
int[] vectorNumeros3 = new int[10];
//Forma 4:
int vectorNumeros4[];
//Forma 5:
int vectorNumeros5[] = {};
//Forma 6:
int vectorNumeros6[] = {9,8,7,6,5,4,3,2,1,0};
//Forma 7:
int vectorNumeros7[] = new int[]{9,8,7,6,5,4,3,2,1,0};
```

CHOUCAIR
Business Centric Testing

# Vectores y matrices

## Ejercicio

Desarrollar un programa que, mediante el uso de dos vectores, permita almacenar las 5 notas de un estudiante para calcular su promedio final, los valores para calcular el promedio se encuentran en un vector a parte que contiene los porcentajes asignados a cada nota.

| Porcentaje | | | | |
|---|---|---|---|---|
| **Nota 1** | **Nota 2** | **Nota 3** | **Nota 4** | **Nota 5** |
| 20% | 10% | 30% | 20% | 20% |

CHOUCAIR
Business Centric Testing

# Vectores - Solución

```
double notas[] = new double[5];
int porcentajes[] = {20,10,30,20,20};

notas[0] = 4.5;
notas[1] = 3.2;
notas[2] = 5.0;
notas[3] = 1.5;
notas[4] = 4.3;

double promedio = 0;

for(int i = 0; i < notas.length; i++)
{
    promedio = ( notas[i] * (porcentajes[i]) / 100) + promedio;
}

System.out.println("Promedio final " + Math.round(promedio));
```

CHOUCAIR
Business Centric Testing

# Vectores y matrices

## Matrices

Las matrices son una estructura de datos que permite almacenar un grupo de datos de un mismo tipo al igual que ocurre con los vectores, pero de una forma bidimensional de forma que se representan como una tabla con un vector para filas y otro para columnas.

*Tomado de la página de campusmvp.es*

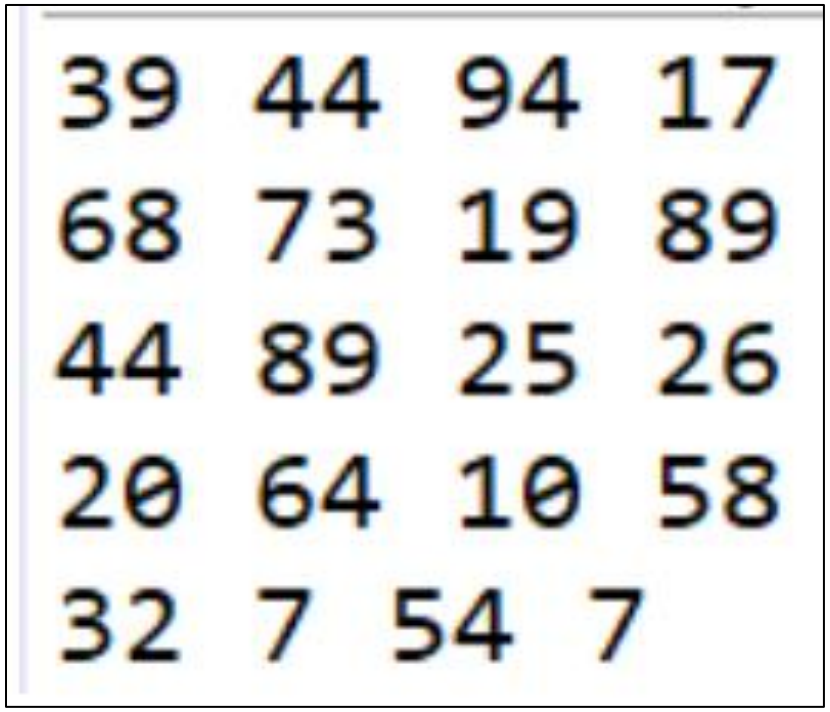
```
//Forma 1:
int matrizNumeros1[][] = new int[4][5];
//Forma 2:
int [][]matrizNumeros2 = new int[4][5];
//Forma 3:
int[][] matrizNumeros3 = new int[4][5];
//Forma 4:
int matrizNumeros4[][];
//Forma 5:
int matrizNumeros5[][] = {};
//Forma 6:
int matrizNumeros6[][] = {{1,2},{3,9}};
//Forma 7:
int matrizNumeros7[][] = new int[][]{{6,2},{2,7}};
```

CHOUCAIR
Business Centric Testing

# Matrices - Ejercicio

Ejercicio: Desarrollar un programa que dada una matriz de 5×4, almacene números aleatorios entre 0 y 100 en todas sus posiciones. Luego mostrar la matriz en forma de tabla.

*Tomado de la página de campusmvp.es*



CHOUCAIR
Business Centric Testing

# Vectores y matrices

```
int numeros[][] = new int[5][4];

for(int i = 0; i < 5; i++)
{
    for(int j = 0; j < 4; j++)
    {
        numeros[i][j] = (int) Math.ceil(Math.random()*100);
    }
}

for(int i = 0; i < 5; i++)
{
    for(int j = 0; j < 4; j++)
    {
        System.out.print(numeros[i][j] + " ");
    }
    System.out.println();
}
```

CHOUCAIR
Business Centric Testing

# Listas

El uso de listas en Java es una forma útil de almacenar y manipular grandes volúmenes de datos, tal como se haría en una matriz o arreglo, pero con una serie de ventajas que hacen de este tipo de variables las preferidas para el procesamiento de grandes cantidades de información.

La implementación de las listas en Java, al igual que otras estructuras de datos se puede realizar de dos formas diferentes:

- Clases e Interfaces de Java.
- Implementación desde cero.

Las estructuras más usadas son:

- List
- ArrayList
- LinkedList

CHOUCAIR
Business Centric Testing

# Listas: Lists

Para el uso de clases e interfaces de este tipo es necesario hacer la respectiva importación de los paquetes de Java que se desean trabajar. List opera con dos paquetes en este caso: List y ArrayList.

```
package Clase;

import java.util.ArrayList;
import java.util.List;

public class Main {

    public static void main(String[] args)
    {
        List<String> listaNombres = new ArrayList<String>();

    }
}
```

EJERCICIO PRÁCTICO

**Ejercicio con listas**

Crea un programa que maneje una lista de nombres. El programa debe agregar algunos nombres a la lista, mostrar todos los nombres en pantalla, mostrar el primer nombre de la lista y luego eliminar uno de los nombres para mostrar cómo queda la lista después de la eliminación.

*Tomado de la página de campusmvp.es*

CHOUCAIR
Business Centric Testing

# Listas: Ejemplos de Código

- List de Objetos.

```
public static void main(String[] args)
{
    List<Object> lista = new ArrayList<Object>();
}
```

- List de un tipo de clase.

```
public static void main(String[] args)
{
    List<Usuario> lista = new ArrayList<Usuario>();
}
```

- List de enteros.

```
public static void main(String[] args)
{
    List<Integer> lista = new ArrayList<Integer>();

}
```

- List de Doubles.

```
public static void main(String[] args)
{
    List<Double> lista = new ArrayList<Double>();

}
```

- List de cualquier tipo.

```
public static void main(String[] args)
{
    List lista = new ArrayList();

}
```

- List con tamaño establecido.

```
public static void main(String[] args)
{
    List<String> lista = new ArrayList<String>(10);
}
```

CHOUCAIR
Business Centric Testing

# Métodos List - add

## add

```
public static void main(String[] args)
{
    List<String> lista = new ArrayList<String>();

    lista.add("Colombia");
    lista.add("Chile");
    lista.add("Argentina");
    lista.add("Venezuela");
    lista.add("Perú");

}
```

## add con índice

```
public static void main(String[] args)
{
    List<String> lista = new ArrayList<String>();

    lista.add(0, "Colombia");
    lista.add(2, "Chile");
    lista.add(3, "Argentina");
    lista.add(5, "Venezuela");
    lista.add(10, "Perú");

}
```

CHOUCAIR
Business Centric Testing

# Métodos clase List

## Add List

```
public static void main(String[] args)
{
    List<String> lista = new ArrayList<String>();
    lista.add(0, "Colombia");
    lista.add(1, "Chile");
    lista.add(2, "Argentina");
    lista.add(3, "Venezuela");
    lista.add(4, "Perú");

    List<String> listaNueva = new ArrayList<String>();

    listaNueva.add("México");
    listaNueva.add("Panamá");
    listaNueva.add("Ecuador");

    lista.addAll(listaNueva);

}
```

## Add List con índice

```
public static void main(String[] args)
{
    List<String> lista = new ArrayList<String>();
    lista.add(0, "Colombia");
    lista.add(1, "Chile");
    lista.add(2, "Argentina");
    lista.add(3, "Venezuela");
    lista.add(4, "Perú");

    List<String> listaNueva = new ArrayList<String>();

    listaNueva.add("México");
    listaNueva.add("Panamá");
    listaNueva.add("Ecuador");

    lista.addAll(5, listaNueva);

}
```

CHOUCAIR
Business Centric Testing

# Métodos clase List

## set

```
public static void main(String[] args)
{
    List<String> lista = new ArrayList<String>();
    lista.add(0, "Colombia");
    lista.add(1, "Chile");
    lista.add(2, "Argentina");
    lista.add(3, "Venezuela");
    lista.add(4, "Perú");

    lista.set(0, "Costa Rica");
}
```

## get

```
public static void main(String[] args)
{
    List<String> lista = new ArrayList<String>();
    lista.add(0, "Colombia");

    System.out.println(lista.get(0));
}
```

CHOUCAIR
Business Centric Testing

# Métodos clase List

## size

```
public static void main(String[] args)
{
    List<String> lista = new ArrayList<String>();

    lista.add(0, "Colombia");
    lista.add(1, "Ecuador");
    lista.add(2, "Venezuela");
    lista.add(3, "Argentina");

    System.out.println("El tamaño de la lista es: " + lista.size());
}
```

## contains

```
public static void main(String[] args)
{
    List<String> lista = new ArrayList<String>();

    lista.add(0, "Colombia");
    lista.add(1, "Ecuador");
    lista.add(2, "Venezuela");
    lista.add(3, "Argentina");

    System.out.println("La lista contiene a Colombia: " + lista.contains("Colombia"));
}
```

CHOUCAIR
Business Centric Testing

# Métodos clase List

## clear

```
public static void main(String[] args)
{
    List<String> lista = new ArrayList<String>();

    lista.add(0, "Colombia");
    lista.add(1, "Ecuador");
    lista.add(2, "Venezuela");
    lista.add(3, "Argentina");

    lista.clear();

    System.out.println("El tamaño de la lista es: " + lista.size());
}
```

## isEmpty

```
public static void main(String[] args)
{
    List<String> lista = new ArrayList<String>();

    lista.add(0, "Colombia");
    lista.add(1, "Ecuador");
    lista.add(2, "Venezuela");
    lista.add(3, "Argentina");

    if(lista.isEmpty())
    {
        System.out.println("La lista está vacía");
    }
    else
    {
        System.out.println("La lista no está vacía");
    }
}
```

CHOUCAIR
Business Centric Testing

# Métodos clase list

## Remove por índice

```
public static void main(String[] args)
{
    List<String> lista = new ArrayList<String>();

    lista.add(0, "Colombia");
    lista.add(1, "Ecuador");
    lista.add(2, "Venezuela");
    lista.add(3, "Argentina");

    lista.remove(3);

    System.out.println(lista.get(3));
}
```

## Remove por valor

```
public static void main(String[] args)
{
    List<String> lista = new ArrayList<String>();

    lista.add(0, "Colombia");
    lista.add(1, "Ecuador");
    lista.add(2, "Venezuela");
    lista.add(3, "Argentina");

    lista.remove("Argentina");

    System.out.println(lista.get(3));
}
```

CHOUCAIR
Business Centric Testing

# Métodos clase list

## indexOf

```
public static void main(String[] args)
{
    List<String> lista = new ArrayList<String>();

    lista.add(0, "Colombia");
    lista.add(1, "Ecuador");
    lista.add(2, "Venezuela");
    lista.add(3, "Argentina");


    System.out.println("La posición es: " +lista.indexOf("Venezuela"));
}
```

## iterator

```
public static void main(String[] args)
{
    List<String> lista = new ArrayList<String>();

    lista.add(0, "Colombia");
    lista.add(1, "Ecuador");
    lista.add(2, "Venezuela");
    lista.add(3, "Argentina");

    Iterator<String> listaIterable = lista.iterator();

    while(listaIterable.hasNext())
    {
        System.out.println("Valor: " + listaIterable.next());
    }
}
```

CHOUCAIR
Business Centric Testing

# Excepciones en Java

> En Java, los errores en tiempo de ejecución se denominan excepciones. Ocurren cuando se produce un error en alguna de las instrucciones del programa, mostrando un mensaje de error y finalizando la ejecución.

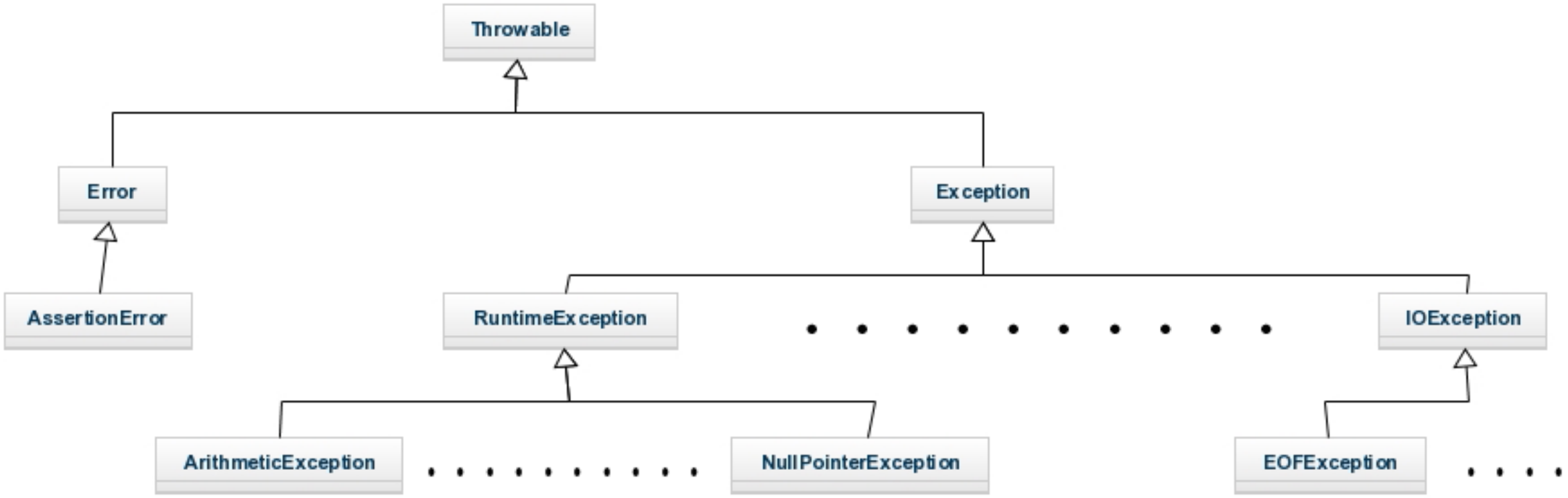
## Ejemplos comunes de excepciones:

- División entre cero
- Objeto con valor 'null' cuando no puede serlo
- Error al abrir un fichero
- Acceso a índice fuera de rango en arrays

Para un mejor manejo de las excepciones se recomienda el uso de try ... catch

CHOUCAIR
Business Centric Testing

# Excepciones: Ejemplo Visual



CHOUCAIR
Business Centric Testing

# Manejo de Excepciones

## Estructura try-catch-finally

```
try {

        // Instrucciones cuando no hay una excepción

} catch (TypeException ex) {

        // Instrucciones cuando se produce una excepcion

} finally {

        // Instruciones que se ejecutan, tanto si hay como sino hay excepciones

}
```

CHOUCAIR
Business Centric Testing

# Manejo de Excepciones: Sin Control de Errores

```
1  package Main;
2
3  public class Main {
4
5      public static int numerador = 10;
6      public static Integer denominador = 0;
7      public static float division;
8
9      public static void main(String[] args) {
10
11     1  System.out.println("ANTES DE HACER LA DIVISIÓN");
12
13     2  division = numerador / denominador;
14
15     3  System.out.println("DESPUES DE HACER LA DIVISIÓN");
16     }
17 }
```

Problems | Javadoc | Declaration | Console

```
<terminated> Main (5) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_20.j
ANTES DE HACER LA DIVISIÓN
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Main.Main.main(Main.java:13)
```

CHOUCAIR
Business Centric Testing

# Manejo de Excepciones - Con Control

```
1  package Main;
2
3  public class Main {
4
5      public static int numerador = 10;
6      public static Integer denominador = 0;
7      public static float division;
8
9      public static void main(String[] args) {
10         System.out.println("ANTES DE HACER LA DIVISIÓN");
11         try {
12             division = numerador / denominador;
13         } catch (ArithmeticException ex) {
14             division = 0; // Si hay una excepción doy valor '0' al atributo 'division'
15             System.out.println("Error: "+ex.getMessage());
16         } finally {
17             System.out.println("División: "+division);
18             System.out.println("DESPUES DE HACER LA DIVISIÓN");
19         }
20     }
21 }
```

Problems @ Javadoc Declaration Console

```
<terminated> Main (5) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_20.jdk/Contents/Home/bin/java (1
ANTES DE HACER LA DIVISIÓN
Error: / by zero
División: 0.0
DESPUES DE HACER LA DIVISIÓN
```

CHOUCAIR
Business Centric Testing

# Estándar de Codificación Java

- Nomenclatura: El idioma por defecto es una mezcla entre nomenclatura tradicional en inglés y nomenclatura funcional adoptada.

  Ejemplos: insert, update, delete, create, retrieve, list, set, get, newInstance, Delegate

- Paquetes: Por defecto todos los paquetes se escribirán en minúsculas y sin utilizar caracteres especiales.

  El paquete base queda definido como es.gobcantabria, en este paquete no se definirá ninguna clase.

- Interfaces: Los nombres de interfaces deben seguir convenciones claras.

  Ejemplos: ConexionInterface, ComponenteTablaInterface

- bussines
  - dao
  - domain
  - service
  - helper
  - exception
- util
- web
  - controller
  - model
  - view

CHOUCAIR
Business Centric Testing

# Estándar de Codificación Java

## Nombres de Clases

- Deben usar CamelCase con la primera letra de cada palabra en mayúsculas
- Mantener los nombres simples y descriptivos
- Usar palabras completas y evitar abreviaturas
- Se permiten acrónimos comunes: DAO, DTO, URL, HTML

| Paquete | Funcionalidad | Nombre |
|---|---|---|
| bussines.dao | Data Access Object (Interface) | <nombre> DAO |
| bussines.dao.impl | Data Access Object (Implementation) | <nombre> DAOImpl |
| bussines.exception | Excepciones | <nombre>Exception |
| bussines.service | Service | <nombre>Service |
| bussines.helper | Helper | <nombre> Helper |
| bussines.dto | Data Transfer Objects | <nombre> DTO |
| util | Clases de constantes. | <scope> Keys<br><nombre> Keys |
| web.controller | Controller | <nombre> Controller |
| web.filter | Filter | <nombre> Filter |
| web.model | Model | <nombre> Model |
| web.listener | Listener | <nombre>Listener |

CHOUCAIR
Business Centric Testing

# Estándar de Codificación Java

## Métodos

- Deben ser verbos en infinitivo
- Usar lowerCamelCase: primera letra en minúscula, primera letra de cada palabra interna en mayúscula
- No se permiten caracteres especiales
- El nombre debe ser suficientemente descriptivo, sin importar la longitud

## Variables

- Mismo tratamiento que los métodos: lowerCamelCase
- Importa la relación entre regla mnemónica y longitud del nombre
- Correctos: `diaCalculo`, `fechaIncorporacion`
  Incorrectos: `dC`, `DCal`, `fI`, `FI`

## Constantes

- Escribirse en MAYÚSCULAS con palabras separadas por guiones bajos `_`
- Todas deben declararse como public static final

# Programación Orientada a Objetos

# Programación Orientada a Objetos

## ¿Qué es la POO?

La programación orientada a objetos es un paradigma de la programación, una propuesta tecnológica adoptada por una comunidad de programadores y desarrolladores cuyo núcleo central es incuestionable en cuanto que únicamente trata de resolver uno o varios problemas claramente delimitados.

Es una forma por la cual los programadores emplean para resolver problemas a partir de clases y objetos, es una forma especial de programar, más cercana a como se expresan las cosas en la vida real.

CHOUCAIR
Business Centric Testing

# Clases

- Las clases son declaraciones de objetos, también se podrían definir como abstracciones de objetos o moldes.
- Esto quiere decir que la definición de un objeto es una clase.
- Cuando se programa un objeto y se definen sus características y funcionalidades, en realidad lo que se hace es programar una clase.

```java
public class Persona {
    String nombre;
    String pais;
    int edad;

    // Constructor correcto
    public Persona() {
        nombre = "Diego";
        pais = "Colombia";
        edad = 22;
    }

    // Método saludar
    public void saludar() {
        System.out.println("Hola, soy " + nombre + " de " + pais);
    }
}
```

CHOUCAIR
Business Centric Testing

# Características de una Clase

## Estructura fundamental de una clase en Java:

- Nombre: Identifica la clase de forma única en el proyecto
- Atributos: Referencia los campos y variables de la clase que permiten definir las características. Los atributos hacen el papel de variables en las clases
- Métodos: Definen el comportamiento y las acciones que puede realizar la clase
- Recomendación: Si se inicializa un atributo, se recomienda el uso de la palabra reservada final para determinarse como constante

```
String nombre;
String pais;
int edad;
```

```
final private String raza = "Perro";
private String nombre;
private int edad;
private String encargada;
```

CHOUCAIR
Business Centric Testing

# Getters y Setters

## Métodos de Acceso

Los Setters y Getters son métodos de acceso que proporcionan una interfaz pública para modificar y obtener los valores de los atributos privados de una clase.

Cuando los atributos de una clase son privados, no hay manera de acceder a ellos directamente desde fuera de la clase. Los métodos de acceso son necesarios para interactuar con estos atributos de forma controlada y segura.

CHOUCAIR
Business Centric Testing

# Setters

- Definición: Hace referencia a la acción de establecer, sirve para asignar un valor inicial a un atributo de forma explícita
- El Setter nunca retorna nada (siempre asigna)
- Solo permite dar acceso público a ciertos atributos que el usuario pueda modificar

```
public void setNombre(String nombre) {
    this.nombre = nombre;
}
```

CHOUCAIR
Business Centric Testing

# Getters

- Definición: Hace referencia a la acción de obtener, sirve para obtener (recuperar o acceder) el valor ya asignado a un atributo y ser utilizado.
- Permite recuperar y utilizar el valor de un atributo.

```
public String getNombre() {
    return nombre;
}
```

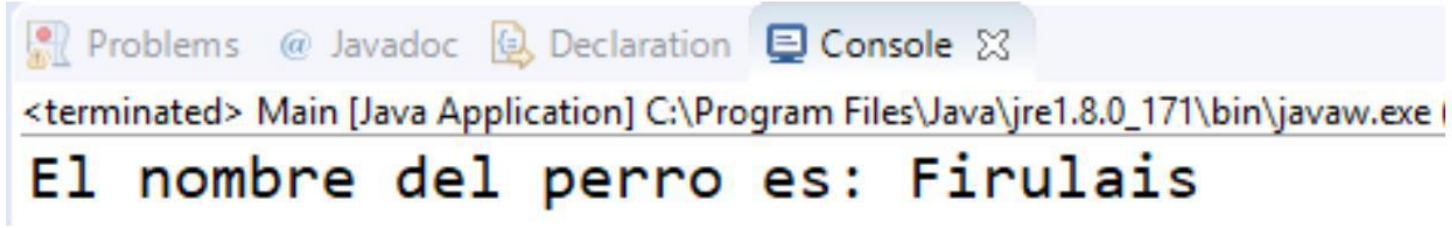
CHOUCAIR
Business Centric Testing

# Instancia

- Se llama instancia a todo objeto que derive de algún otro.
- Es el proceso por el cual se crea un ejemplar (instancia) de un objeto a partir de la definición de la clase.

```
public static void main(String[] args)
{
    Perro Firulais = new Perro();

    Firulais.setNombre("Firulais");
    System.out.println("El nombre del perro es: " + Firulais.getNombre());
}
```



CHOUCAIR
Business Centric Testing

# This

- Definición: Sirve para hacer referencia a un método, propiedad o atributo del objeto actual
- Uso principal: Se utiliza principalmente cuando existe sobrecarga de nombres
- Sobrecarga de nombres: Se da cuando hay una variable local de un método o constructor, o un parámetro formal de un método o constructor, con un nombre idéntico al que está presente en la clase en el momento de relacionarse

```
public Perro(String nombre, int edad, String encargada) {
    this.nombre = nombre;
    this.edad = edad;
    this.encargada = encargada;
}
```

CHOUCAIR
Business Centric Testing

# Parámetros o Argumentos

- Los parámetros o argumentos son una forma de intercambiar información con el método.
- Pueden servir para introducir datos para ejecutar el método (entrada).
- O para obtener o modificar datos tras su ejecución (salida).

Ejemplo:

```
miCarro.acelerar();
```

miCarro = objeto

CHOUCAIR
Business Centric Testing

# Parámetros vs Argumentos

## Parámetros

Son los valores que un método recibe desde un objeto.

Declaración del Método

```
//Constructor Parametros que debe recibir el método constructor
  public Carro(String marca, String modelo, String color, boolean enVenta) {
      this.marca = marca;
      this.modelo = modelo;
      this.color = color;
      this.enVenta = enVenta;
  }
```

## Argumentos

Son los valores que un objeto recibe para operar un método.

Uso del Método

```
public static void main(String[] args)
{
  //Clase Objeto                    Argumentos al método constructor
    Carro Tracker = new Carro("Chevrolet", "Negro", "Tracker LT", false);
}
```

CHOUCAIR
Business Centric Testing

# Constructores

- Es un método que contiene las acciones que se realizarán por defecto al crear un objeto, en la mayoría de los casos, se inicializan los valores de los atributos en el constructor.
- No es obligatoria su creación.
- El nombre debe ser el mismo de la clase.
- No retorna ningún valor.
- Pueden existir varios constructores (sólo se deben diferenciar en sus parámetros).
- Dado el caso de existir varios constructores, sólo se ejecutará uno de éstos.
- Se recomienda el uso de modificadores de accesos.

CHOUCAIR
Business Centric Testing

# Constructores: Ejemplo

## Declaración del Constructor

```
public Perro(String nombre, int edad, String encargada) {
    this.nombre = nombre;
    this.edad = edad;
    this.encargada = encargada;
}
```

## Uso del Constructor

```
public static void main(String[] args)
{
    Perro Firulais = new Perro("Firulas", 9, "Ana");
}
```

CHOUCAIR
Business Centric Testing

# Sobrecarga de Constructores

Permite definir más de un constructor con el mismo nombre, con la condición de que no puede haber dos de ellos con el mismo número y tipo de parámetros.

```
public Perro()
{

}

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

public Perro(String nombre)
{
    this.nombre = nombre;
}


   public static void main(String[] args)
   {
       Perro Firulais = new Perro();
       Perro Peluche  = new Perro("Peluche", 9, "Yulied");
       Perro Poseidon = new Perro("Poseidon");
   }
```

CHOUCAIR
Business Centric Testing

# Métodos: Definición y Características

- Definición: Procesos o acciones disponibles para el objeto. Abstracción de operaciones que pueden realizarse con un objeto.
- Reutilización: Permiten reutilizar código de manera eficiente.
- Retorno: Pueden o no retornar valores. Los métodos void no retornan, otros retornan tipos específicos (int, String, etc.).
- Parámetros: Pueden contener N parámetros, aunque se recomienda no sobrecargar los métodos.
- Nomenclatura: Los nombres deben ser muy claros y descriptivos con la acción que van a realizar.
- Modificadores: Se recomienda el uso de modificadores de acceso, especialmente diseñar los métodos con el modificador private.
- Estructura: Modificador de acceso + Tipo de dato + Nombre de método + Parámetros

```
public class Casa {

    private String color;
    private int cuartos;
    private int habitantes;
    private String ciudad;
    private int precio;
    private String propietario;
```

CHOUCAIR
Business Centric Testing

# Métodos Void

La utilidad de los métodos void radica en que son métodos que no cuentan con ningún tipo de retorno.

Algunas características de este tipo de método son:

- Se centran en realizar acciones que no requieren retornar un valor en específico, también suele ser usado para mostrar mensajes.
- Se caracterizan por no tener un tipo de dato asociado.
- Siempre contienen la palabra void.
- El modificador de acceso más común es public.
- Puede o no recibir parámetros.
- Los nombres deben ser muy claros y descriptivos con la acción que van a realizar.

```
    public void pintarDeBlanco()
    {
        color = "Blanco";
    }

public static void main(String[] args)
{
    Casa miCasa = new Casa();
    miCasa.setColor("Verde");
    miCasa.pintarDeBlanco();
    System.out.println(miCasa.getColor());
}
```

CHOUCAIR
Business Centric Testing

# Métodos de Tipo

La utilidad de los métodos de tipo radica en que son métodos que cuentan un retorno en base al tipo de dato que fue declarado.

Algunas características de este tipo de método son:

- Se centran en realizar acciones que deben contar con un retorno obligatorio en base al tipo de dato.
- Se caracterizan por tener un tipo de dato asociado.
- El modificador de acceso más común es public.
- Puede o no recibir parámetros.
- Los métodos get son un ejemplo de métodos de tipo.
- Los nombres deben ser muy claros y descriptivos con la acción que van a realizar.

```
public int aumentarPrecio(int precio)
{
    return this.precio = this.precio + precio;
}

    public static void main(String[] args)
    {
        Casa miCasa = new Casa();
        miCasa.setPrecio(100000);
        miCasa.aumentarPrecio(150000);
        System.out.println(miCasa.getPrecio());
    }
```

CHOUCAIR
Business Centric Testing

# Modificadores de Acceso

- Introducen el concepto de encapsulamiento
- Controlan el acceso a los datos que conforman un objeto o instancia
- Dan un nivel de seguridad mayor restringiendo el acceso a diferentes atributos, métodos y constructores
- Aseguran que el usuario deba seguir una "ruta" especificada para acceder a la información
- Aseguran que un valor no será modificado incorrectamente
- El acceso a los atributos se consigue por medio de los métodos get y set
- Recomendación: Los atributos de una clase deben ser privados y cada atributo debe tener sus propios métodos get y set

```
public class Carro
{

    private String marca;
    private String modelo;
    private String color;
    private boolean enVenta;
}
```

```
public class Carro
{

    private String marca;
    private String modelo;
    private String color;
    private boolean enVenta;

    public String getMarca() {
    public void setMarca(String marca) {

    public String getModelo() {
    public void setModelo(String modelo) {

    public String getColor() {
    public void setColor(String color) {

    public boolean isEnVenta() {
    public void setEnVenta(boolean enVenta) {


}
```

CHOUCAIR
Business Centric Testing

# Modificadores de Acceso - Tipos

- Private: Es el modificador más restrictivo y especifica que los elementos que lo utilizan sólo pueden ser accedidos desde la misma clase en la que se encuentran. Este modificador sólo puede utilizarse sobre los miembros de una clase y sobre interfaces y clases internas, no sobre clases o interfaces de primer nivel.

- Protected: Indica que los elementos sólo pueden ser accedidos desde su mismo paquete y desde cualquier clase que extienda la clase en que se encuentra, independientemente de si esta se encuentra en el mismo paquete o no. Este modificador, como private, no tiene sentido a nivel de clases o interfaces no internas.

- Public: Este nivel de acceso permite acceder al elemento desde cualquier clase, independientemente de que esta pertenezca o no al paquete en que se encuentra el elemento.

- Default: Si no se determina ningún modificador, se usa el de por defecto, que sólo puede ser accedido por clases que están en el mismo paquete.

| Visibilidad | Public | Private | Protected | Default |
|---|---|---|---|---|
| Desde la misma clase | Si | Si | Si | Si |
| Desde cualquier clase del mismo paquete | Si | No | Si | Si |
| Desde una subclase del mismo paquete | Si | No | Si | Si |
| Desde cualquier clase fuera del paquete | Si | No | Si, a través de Herencia | No |
| Desde cualquier subclase fuera del paquete | Si | No | No | No |

CHOUCAIR
Business Centric Testing

# Sobrecarga de Métodos

Permite definir más de un método con el mismo nombre, con la condición de que no puede haber dos de ellos con el mismo número y tipo de parámetros.
El uso de la sobrecarga de métodos se ve sujeto a las siguientes condiciones:

- Deben tener el mismo nombre en los métodos.
- No puede haber dos métodos con el mismo nombre y el mismo tipo con igual número de parámetros, se permite el mismo nombre y mismo tipo siempre y cuando el número de parámetros sea diferente.
- Suele ser utilizada para sobrecarga de métodos con métodos de tipo.
- Java desde la instancia a la hora de usar los métodos los diferenciará en base a los parámetros que éste reciba.

```
public class Calculadora
{
    public int sumar(int numero1, int numero2)
    {
        return numero1+numero2;
    }

    public double sumar(double numero1, double numero2)
    {
        return numero1+numero2;
    }

    public float sumar(float numero1, float numero2)
    {
        return numero1+numero2;
    }

    public int sumar()
    {
        return 0;
    }

    public int sumar(int numero1, int numero2, int numero3)
    {
        return numero1+numero2+numero3;
    }
}

public static void main(String[] args)
{
    Calculadora Operaciones = new Calculadora();

    Operaciones.sumar();
    Operaciones.sumar(9.2, 3.7);
    Operaciones.sumar(9f, 9.5f);
    Operaciones.sumar(5, 2);
    Operaciones.sumar(2, 3, 4);
}
```

CHOUCAIR
Business Centric Testing

# Herencia: Mecanismo de Reutilización

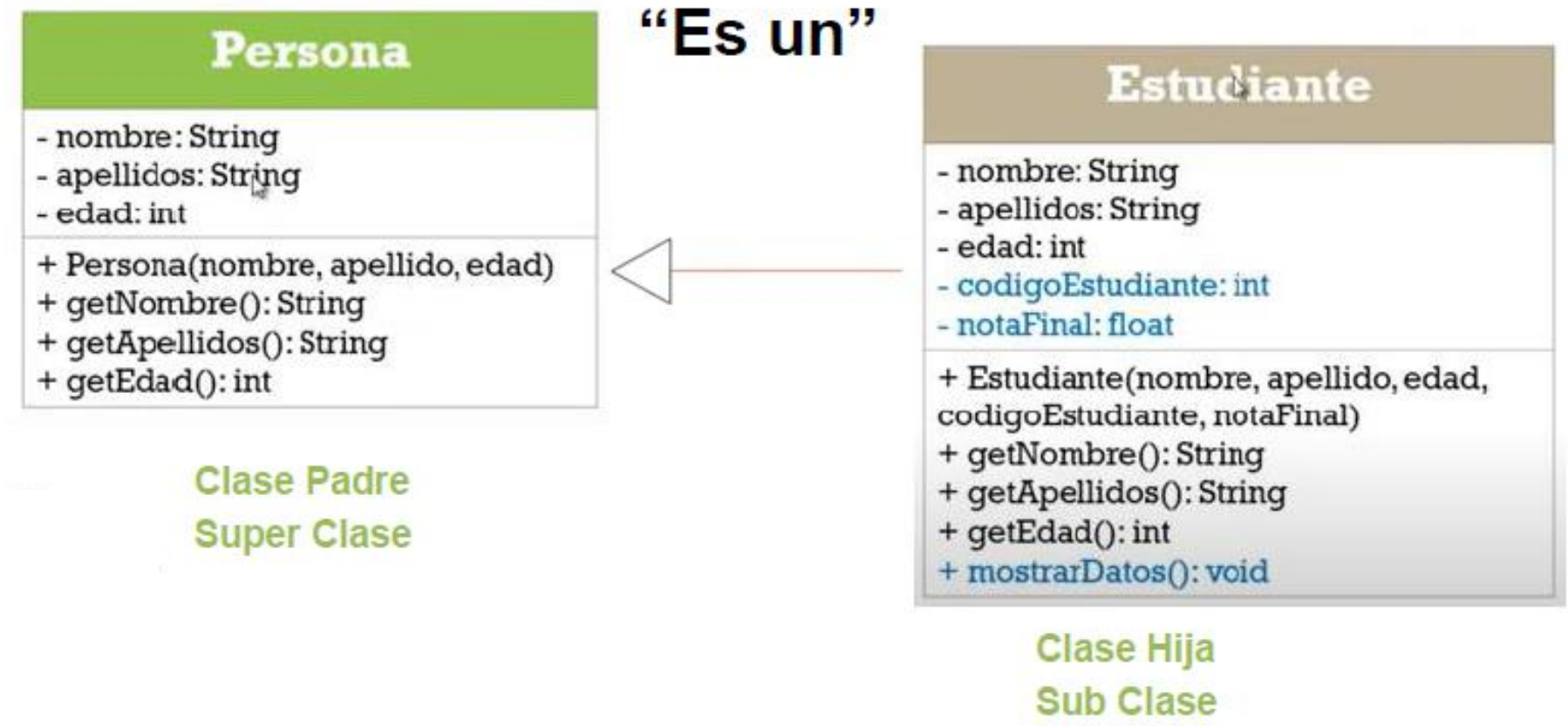
- Definición: Mecanismo que permite la definición de una clase a partir de otra ya existente
- Reutilización: Forma de reutilización de software en la que se crea una nueva clase al absorber los miembros de una ya existente
- Ventaja Principal: Una de las ventajas principales de la Programación Orientada a Objetos es la reutilización de código previamente desarrollado
- Incorporación: Permite a una clase más específica incorporar la estructura y comportamiento de una clase más general

```
package Clase;

public class ClaseB extends ClaseA
{

}
```

# Herencia: Ejemplo




La clase hija hereda las características de la clase padre

CHOUCAIR
Business Centric Testing

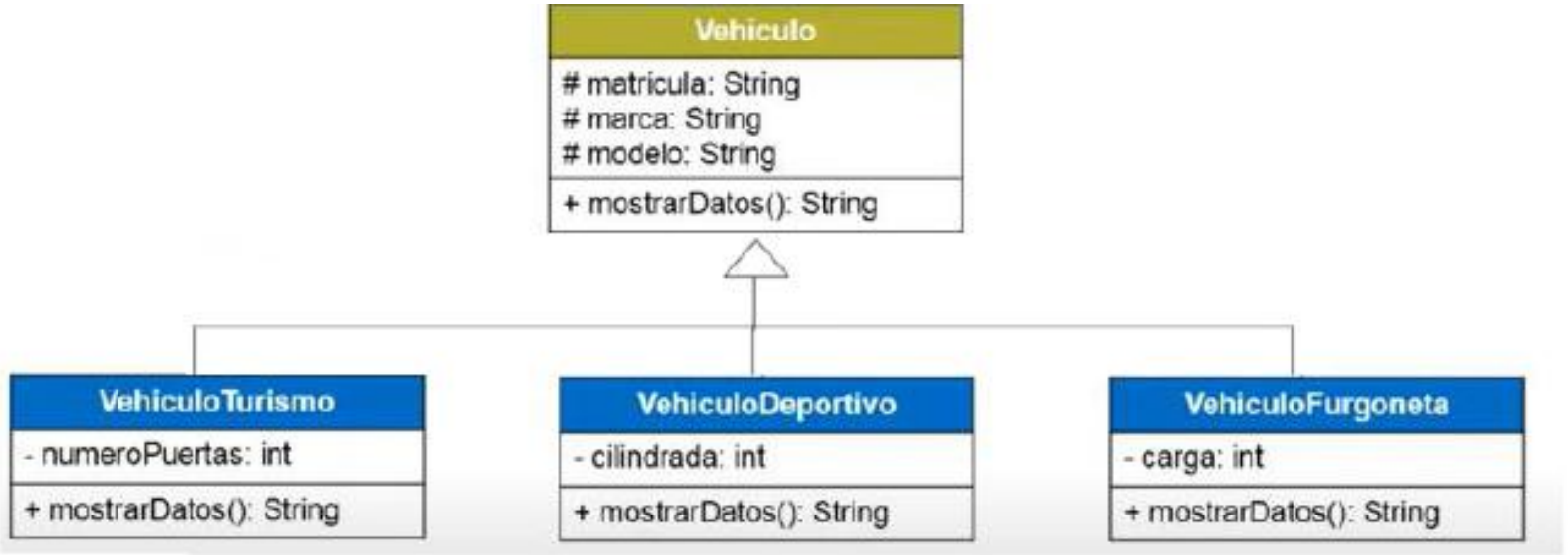
# Polimorfismo: Sobreescritura de Métodos

- Sobreescritura de métodos: Es la forma por la cual una clase que hereda puede re-definir los métodos de su clase Padre
- Permite crear nuevos métodos con el mismo nombre de su superclase
- En una relación de tipo herencia, un objeto de la superclase puede almacenar un objeto de cualquiera de sus subclases
- Esto significa que la clase padre o superclase es compatible con los tipos que derivan de ella. Pero no al revés

| Poli | Morfismo |
| --- | --- |
| "muchos" | "forma" |

CHOUCAIR
Business Centric Testing

# Sobreescritura de Métodos - Polimorfismo




```
Vehiculo miVehiculo = new Vehiculo("12GB","Ferrari","A8");
Vehiculo miVehiculo2 = new VehiculoTurismo("12GB","Ferrari","A8",4);
Vehiculo miVehiculo3 = new VehiculoDeportivo("12GB","Ferrari","A8",500);
```

CHOUCAIR®
Business Centric Testing

# Muchas gracias