UNIVERSIDAD DE LAS FUERZAS ARMADAS "ESPE"

PROG. ORIENTADA A OBJETOS

NRC: 1940
FECHA: 28/08/2024

Programming Paradigms

INTEGRANTES:

-CHASIPANTA IAN

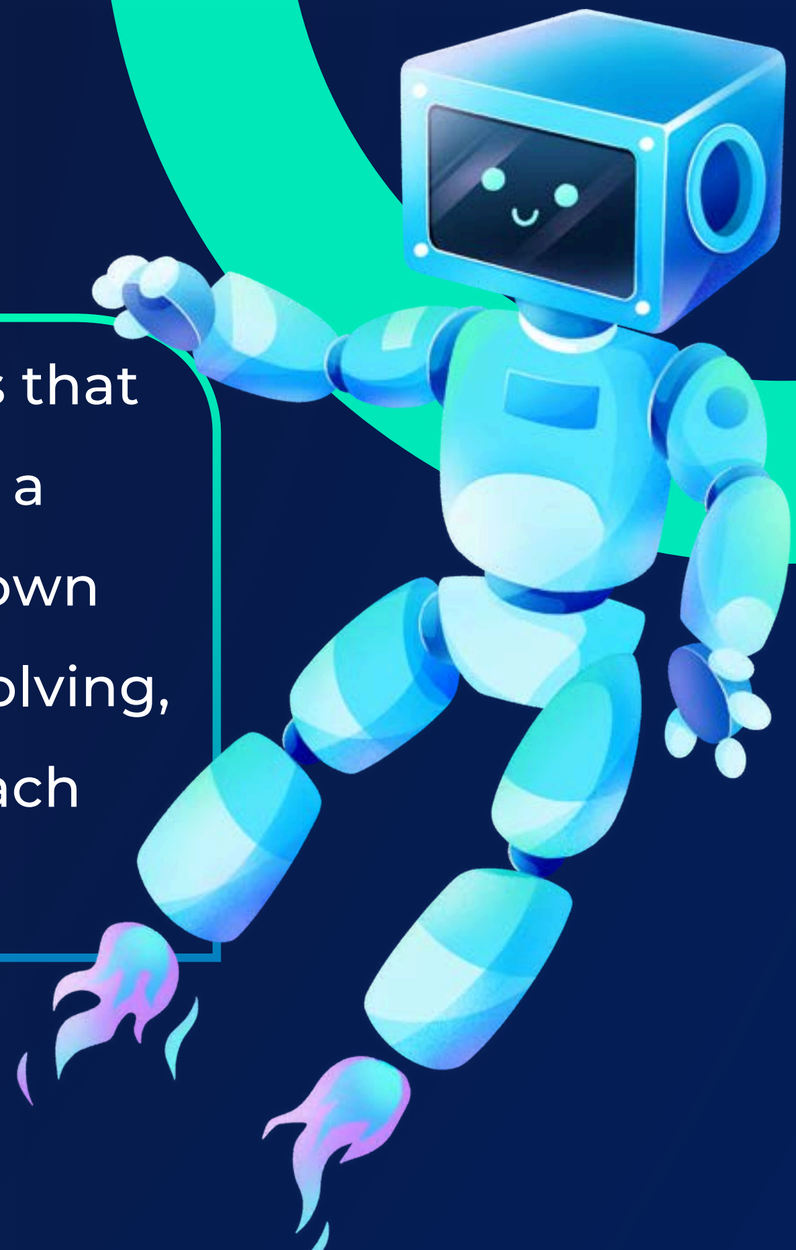
-CALVOPIÑA DAVID

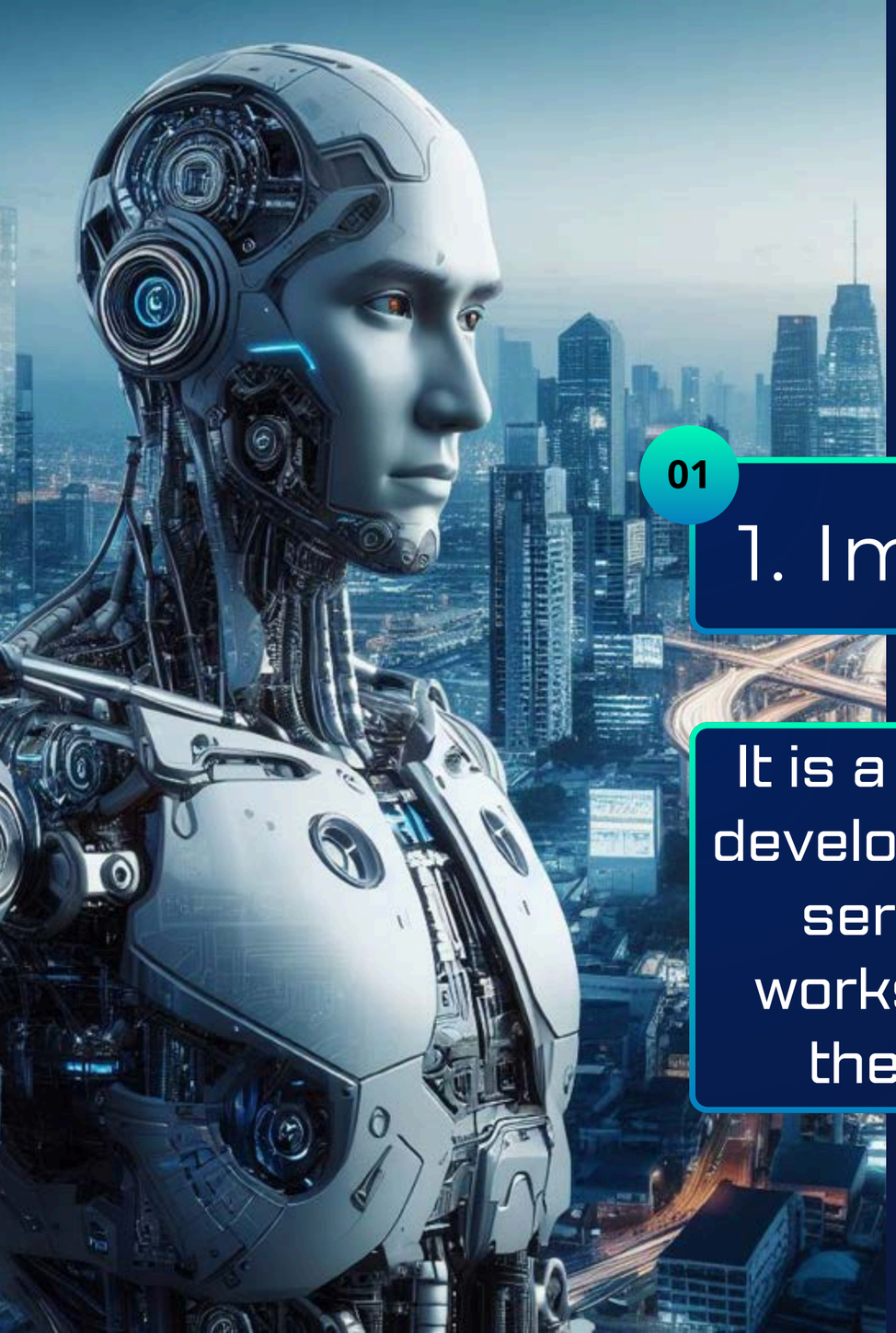
-CHASIPANTA SAUL

-ROJAS JONATHAN

Programming paradigms

Programming paradigms are approaches or styles that guide how code is structured and organized in a programming language. Each paradigm has its own principles, techniques, and methods for problem-solving, influencing how programmers think and approach software development.





The main paradigms in programming

01

1. Imperative Paradigm

It is a method that allows programs to be developed through procedures. Through a series of instructions, how the code works is explained step by step so that the process is as clear as possible.

Example:

C++ Code:

```cpp
/*Implement an algorithm to determine whether
an integer is positive or negative*/

#include <iostream>

using namespace std;

int main() {
    int n1;
    cout << "Enter an integer number:" << endl;
    cin >> n1;

    if (n1 >= 0) {
        cout << "The number is positive." << endl;
    } else {
        cout << "The number is negative" << endl;
    }

    return 0;
}
```

The main paradigms in programming

02

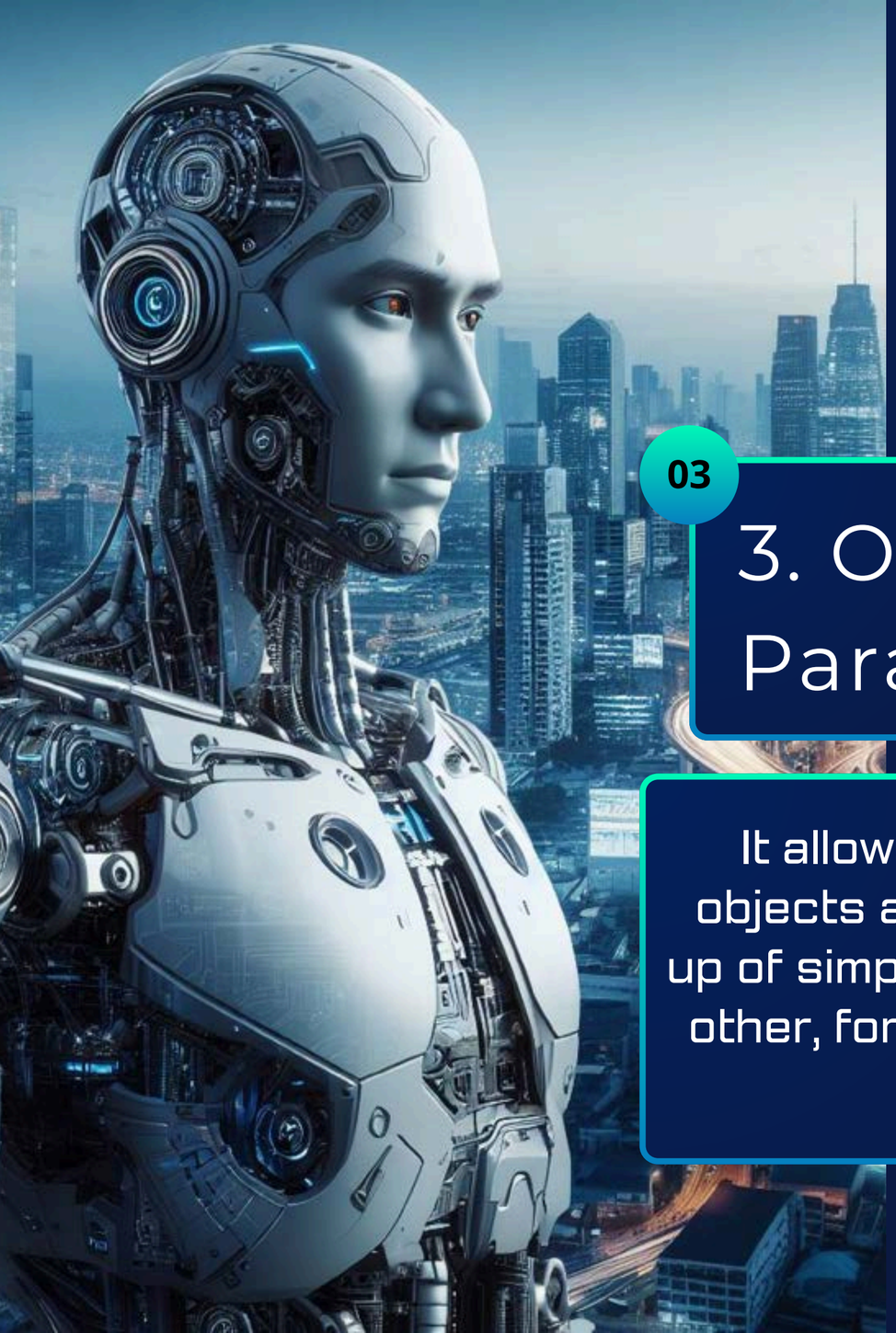
2. Declarative Paradigm

It is concerned with the final result
from the beginning.
The thing that you need / Final result.

Example:

C++ Code:

```cpp
/*Implement an algorithm to display the
odd numbers from 1 to 100.*/

#include <iostream>
using namespace std;

int main() {
    int count=0;
    for (int i=1; i<=100; i+=2) {
        count += 1;
        cout << "Odd # " << count << " is: " << i << endl;
    }
    return 0;
}
```



The main paradigms in programming

03

3. Object Oriented Paradigm (OOP)

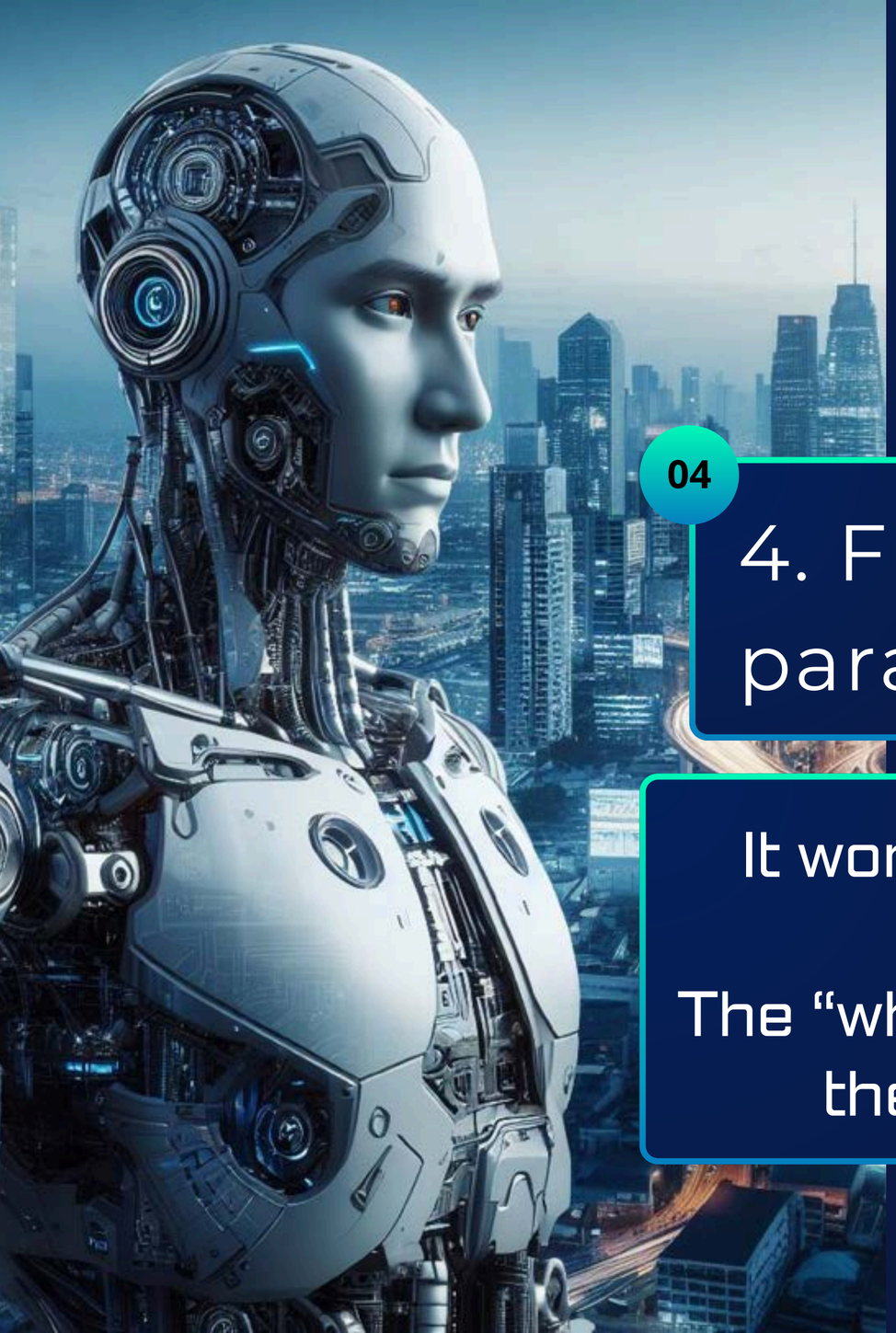
It allows you to identify how to work with it through objects and code plans. This type of paradigm is made up of simple pieces or objects that, when related to each other, form different components of the system we are working on.

Example:
Java

```java
public class GFG {

    static String Employee_name;
    static float Employee_salary;

    static void set(String n, float p) {
        Employee_name  = n;
        Employee_salary  = p;
    }

    static void get() {
        System.out.println("Employee name is: " +Employee_name );
        System.out.println("Employee CTC is: " + Employee_salary);
    }

    public static void main(String args[]) {
        GFG.set("Rathod Avinash", 10000.0f);
        GFG.get();
    }
}
```

Output

```
Employee name is: Rathod Avinash
Employee CTC is: 10000.0
```



The main paradigms in programming

04

4. Functional paradigm

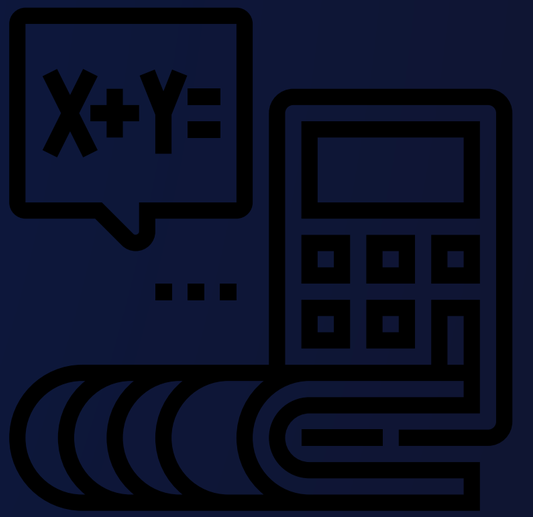
It works through certain mathematical functions.
The “what” matters more and not so much the “how” a project is developed.

Example:
Python

```python
# Implement an algorithm to display the odd numbers from 1 to 100.
numbers = [1, 2, 3, 4, 5]
squares = list(map(lambda x: x ** 2, numbers))
print(squares)
```

```
[1, 4, 9, 16, 25]


...Program finished with exit code 0
Press ENTER to exit console.
```

The main paradigms in programming



05

5. Reactive paradigm

The reactive paradigm is focused on analyzing the flow of data, whether finite or infinite, in order to respond to the needs that arise during the development of projects in order to the change of values, that are produced by data flows.

Example:

Java

```java
/* Create an Observable that emits workdays from Monday to Friday. */

Observable<String> workdays = Observable.fromArray( "Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
workdays.subscribe(
  day -> System.out.println(day),
  error -> System.out.println("Error: " + error),
  () -> System.out.println("Stream completed.")
);
```

Paradigm transition

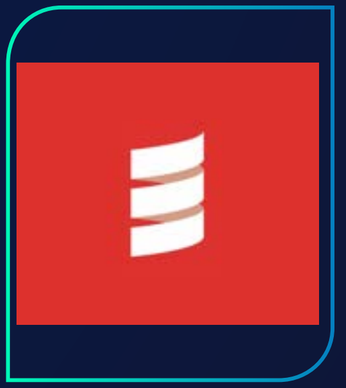
In the world of programming there are different techniques and approaches with which efficient software can be developed in solving problems. These approaches are known as programming paradigms and each of these paradigms has its own style and restrictions, which is why it is They have created different types of paradigms.
Once the usefulness of each paradigm is known, a transition can be made, taking into account the following aspects.
1) Have a solid understanding of the paradigm in which you are going to work
2) Become familiar with the tools used in each paradigm
3) Make the transition gradually

JAVA

PYTHON

C++

SQL



SCALA

C

THANK YOU <3

NO QUESTIONS

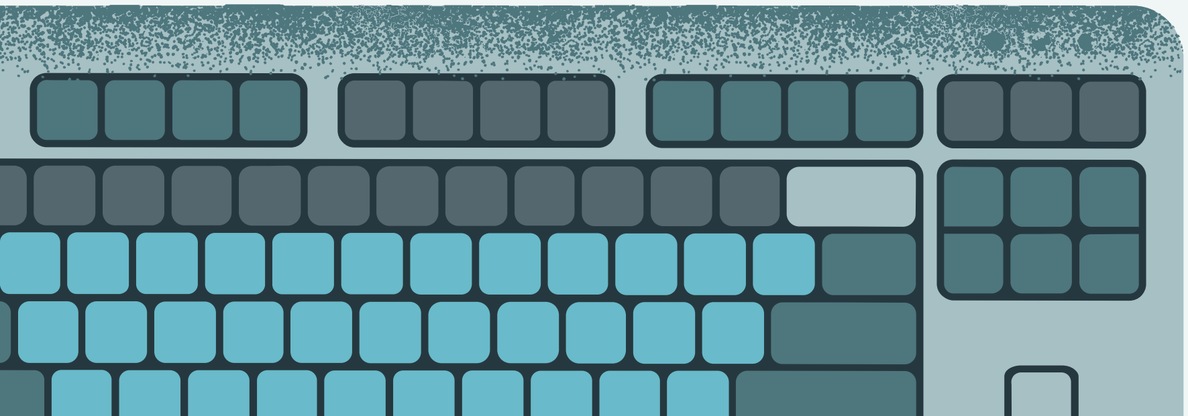
pls

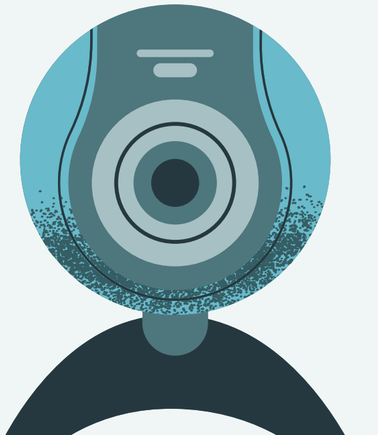
Página de Recursos

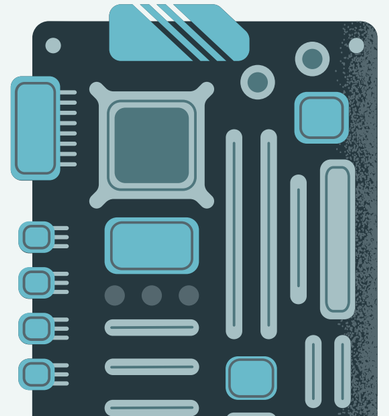
UNIVERSIDAD DE LAS FUERZAS ARMADAS "ESPE"

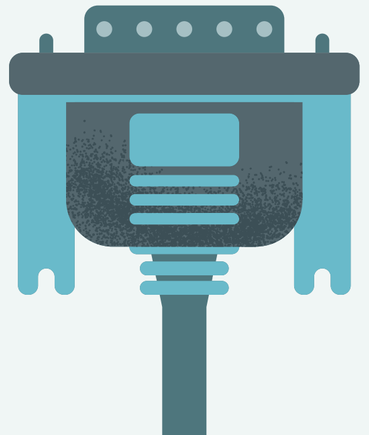
OBJECT-ORIENTED PROGRAMMING

NRC: 1940
DATE: 05/11/2024

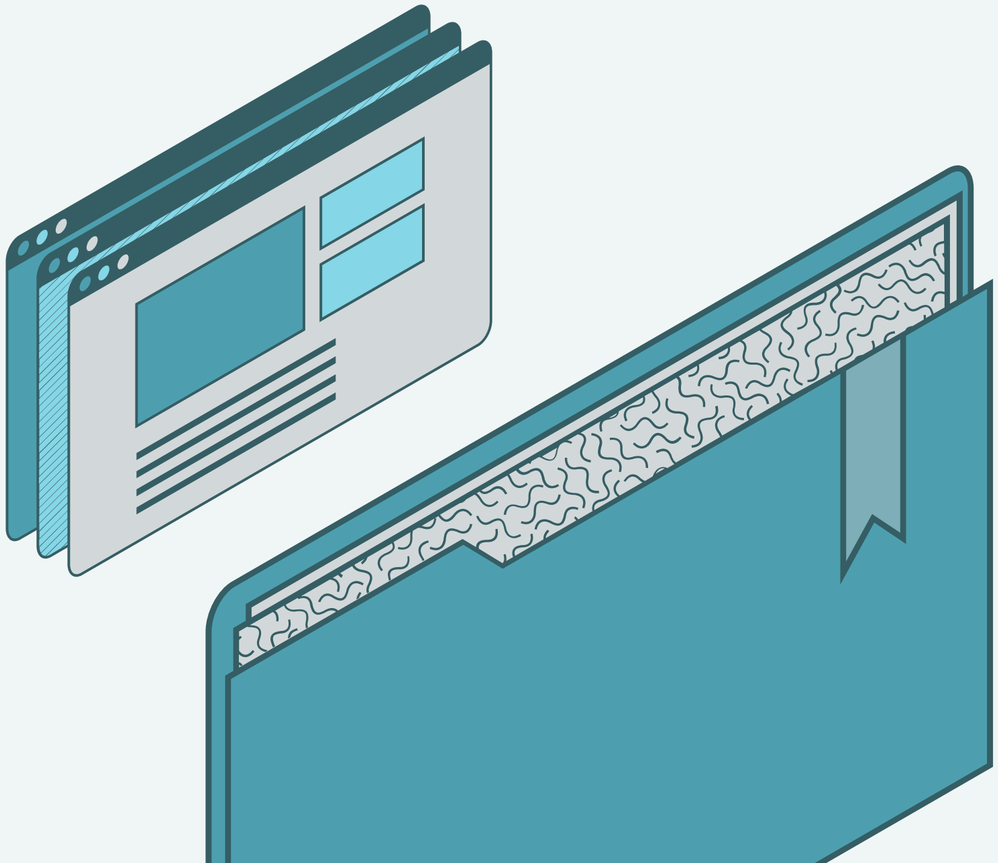








GENERAL PRINCIPLES OF OBJECT-ORIENTED PROGRAMMING



Group Members:

-CHASIPANTA IAN

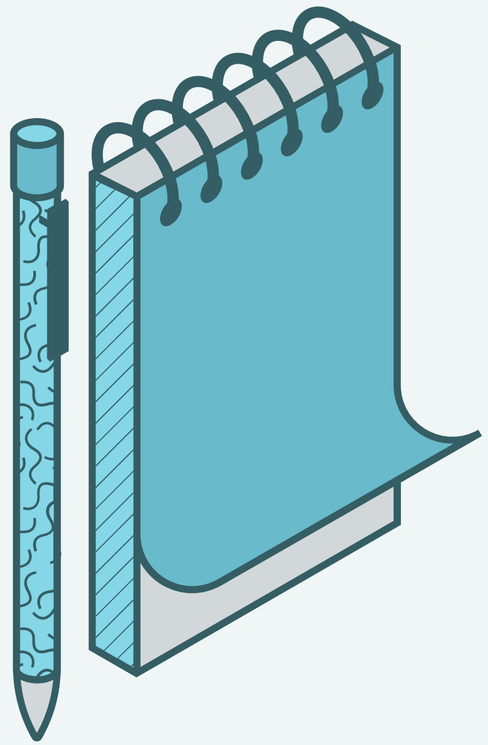
-CALVOPIÑA DAVID

-CHASIPANTA SAUL

-ROJAS JONATHAN

GENERAL PRINCIPLES OF OBJECT-ORIENTED PROGRAMMING

OBJECT-ORIENTED PROGRAMMING (OOP) IS A PARADIGM THAT ORGANIZES SOFTWARE DESIGN AROUND "OBJECTS" RATHER THAN ACTIONS OR FUNCTIONS. THE BASIC PRINCIPLES OF OOP MAKE IT EASIER TO CREATE MORE MODULAR, REUSABLE, AND MAINTAINABLE APPLICATIONS. THE ESSENTIAL CONCEPTS ARE DESCRIBED BELOW:



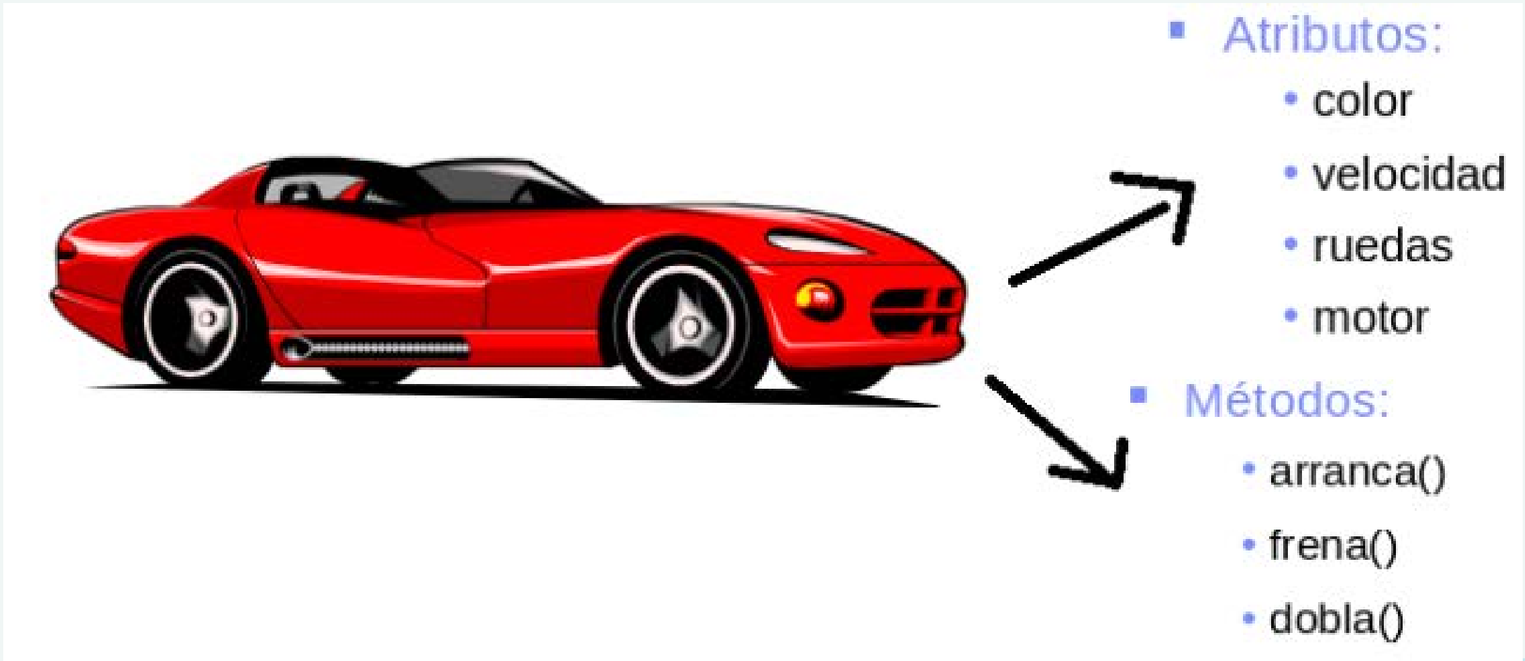
1. CLASSES
-A CLASS IS A TEMPLATE OR MODEL THAT DEFINES A SET OF ATTRIBUTES AND METHODS THAT CHARACTERIZE A PARTICULAR TYPE OF OBJECT.

-CLASSES HELP STRUCTURE THE CODE, SPECIFYING HOW OBJECTS BASED ON THEM SHOULD BEHAVE.

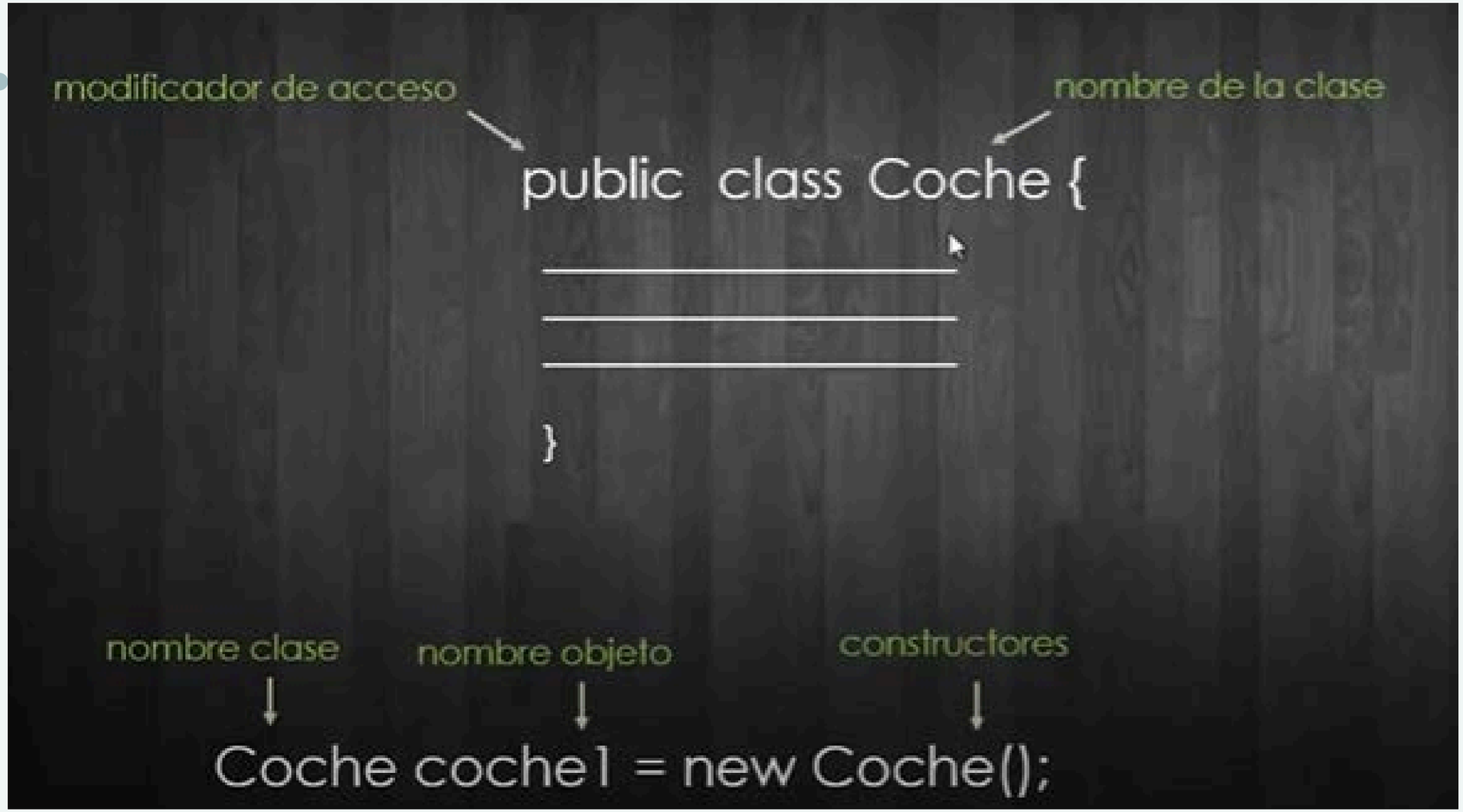
-FOR EXAMPLE, THE VEHICLE CLASS SERVES AS A GENERAL MODEL FOR ANY TYPE OF VEHICLE.

OBJECT

An object, also known as an instance of a class, is the concrete and specific representation of that class that resides in the computer's memory.



CREATING CLASSES AND OBJECTS

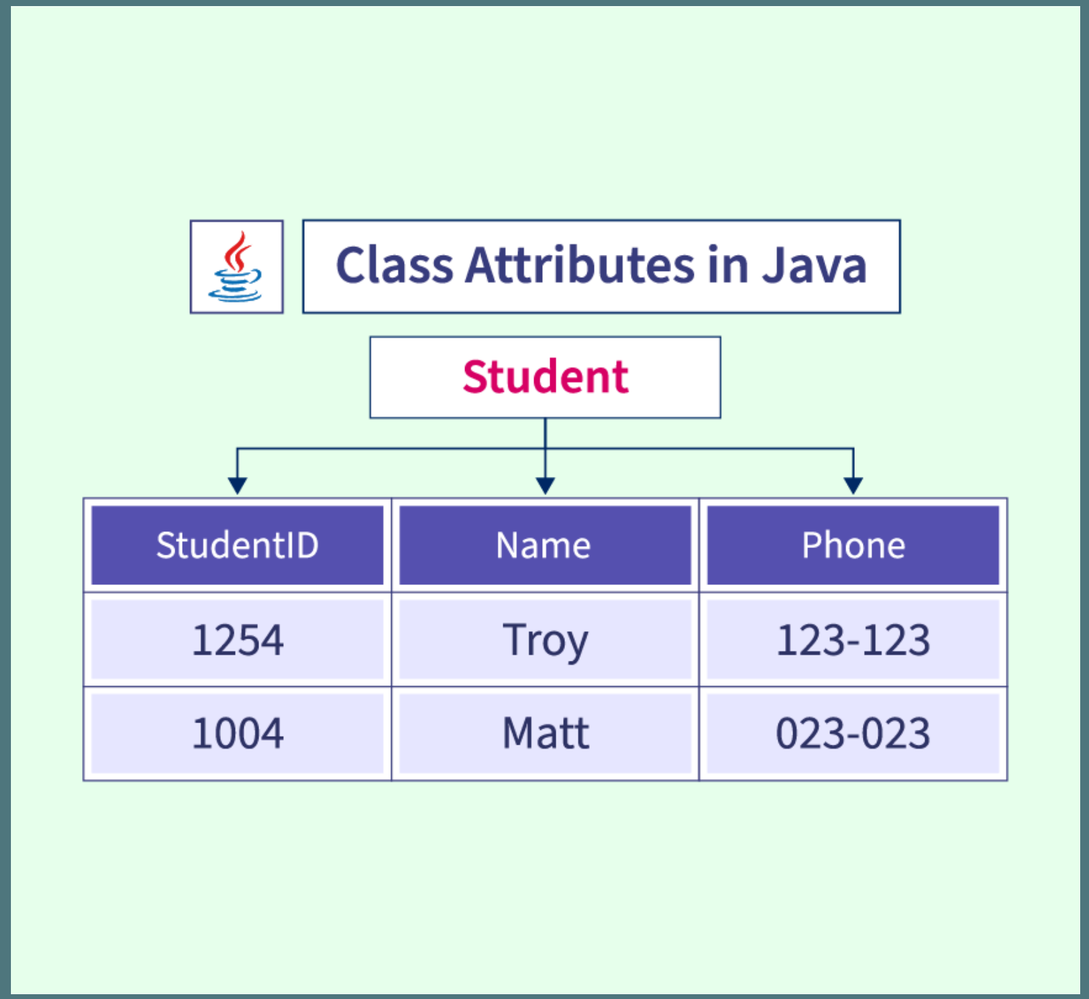


ATTRIBUTES

- Attributes are data members inside a class or object.
- They represent the features or characteristics of the class.
- Example: A car class might include attributes like:
- Tire, door, seats, license plate, headlight, wheel, handle, make, year.
- Defining attributes keeps the code simple and maintainable.

Attributes

>>> Door
>>> Seats
>>> Licence plate
>>> Wheel
>>> Side mirror
>>> Handle
>>> Make



Instance Attribute: Data member of an object, defined inside the constructor. Scope of access is within the object’s creation.

Class Attribute: Defined outside the constructor. Shared and accessible by all objects of the class.

```java
public class Coche {
    //Atributos
    String color;
    String marca;
    int km;

    public static void main(String[] args){
        Coche coche1= new Coche ();
        coche1.color="Azul";
        coche1.marca="Mazda";
        coche1.km=0;

        System.out.println("Color coche 1: "+coche1.color);
        System.out.println("Marca coche 1: "+coche1.marca);
        System.out.println("Kilometraje coche 1: "+coche1.km);

        // Se pueden crear más objetos
    }
}
```

Output - POO (run)

```
run:
Color coche 1: Azul
Marca coche 1: Mazda
Kilometraje coche 1: 0
BUILD SUCCESSFUL (total time: 0 seconds)
```

METHODS (ACTIONS)

In classes, methods define actions that objects can perform, like starting or stopping. They either manage an attribute or carry out a task, ensuring code is clear, consistent, and easy to reuse. Methods are like functions but are specifically tied to the class, so the same actions apply to all objects created from that class.

Methods

```
>>>headLightOn
>>>Start
>>>Brake
>>>Horn
>>>turnOnAC
>>>startWiper
>>>muffler
```

KEY CONCEPTS OF METHODS

```java
package ClasesObjetos;

import java.util.Scanner;

public class Operacion {
    //Atributos
    int n1,n2,suma;

     // Atributo para el objeto Scanner
    Scanner scanner = new Scanner(System.in);

    //Métodos

    //Método para pedir al usuario q nos digite dos números

    public void leer(){
        System.out.print("Digite el primer numero: ");
        n1 = scanner.nextInt();
        System.out.print("Digite el segundo numero: ");
        n2 = scanner.nextInt();
    }

    //Método para sumar ambos números
    public void sumar(){
        suma = n1+n2;

    }

    public void mostrar (){
        System.out.println("La suma es: "+suma);
    }
}
```

```java
    public static void main (String[] args){
        Operacion op = new  Operacion();

        op.leer();
        op.sumar();
        op.mostrar();

    }
}
```

Output - POO2 (run)

```
run:
Digite el primer numero: 4
Digite el segundo numero: 9
La suma es: 13
BUILD SUCCESSFUL (total time: 10 seconds)
```

UNIVERSIDAD DE LAS FUERZAS ARMADAS "ESPE"

OBJECT-ORIENTED PROGRAMMING

NRC: 1940
DATE: 05/11/2024

1922
ECUADOR
ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

Correct use of identifiers

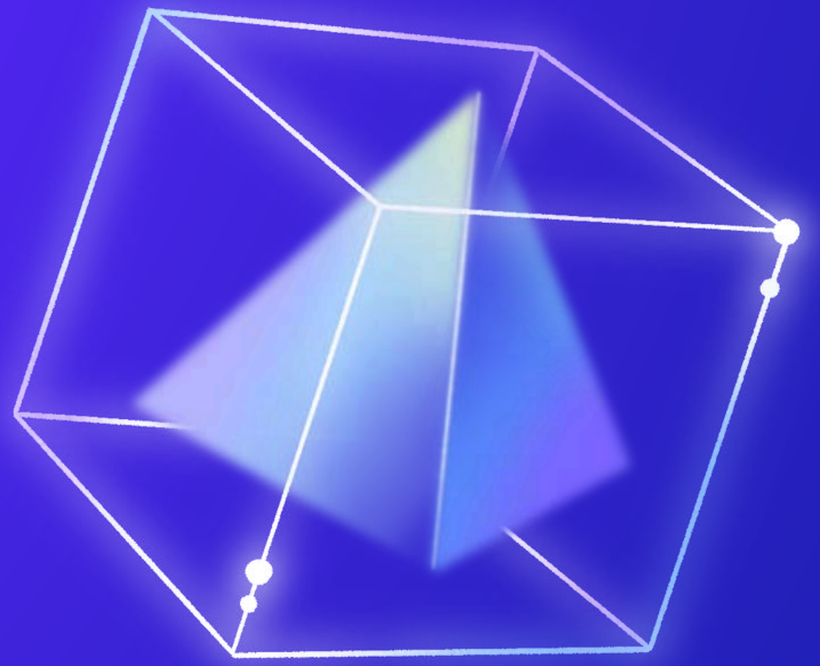
Group Members:

-CHASIPANTA IAN

-CALVOPIÑA DAVID

-CHASIPANTA SAUL

-ROJAS JONATHAN



Correct use of identifiers

In programming, identifiers are the names we assign to variables, functions, classes, methods, etc., to refer to them in the code. Using identifiers correctly is essential to creating code that is clear, easy to understand, and maintain.

Access modifiers

In object-oriented programming, access modifiers are keywords that control the visibility of a class's properties and methods from other classes or outside of it. These modifiers help define access levels to elements of a class, promoting encapsulation and data protection.

The main access modifiers and the implementation of classes in Java

Java has three main access modifiers:

-Public (public): Allows access to attributes or methods from anywhere in the program.

-Private (private): Restricts access only to the class itself. Private attributes and methods cannot be accessed or modified directly from outside the class.

-Protected (protected): Allows access from the same class, subclasses and classes within the same package.

IDENTIFIERS

Class implementation

The class implementation tries to adapt to people's way of thinking and adjust to their way of analyzing.

This stage comprises two components which are the declaration and the body of the class.

ClassDeclaration

{

ClassBody

}

HOW TO...

The class declaration must contain the keyword class and the name of the class that is being defined, for example class imaginarynumber {}

Within the class declaration you can

- Declare what the class is
- List the interfaces implemented by the class
- Declare if the class is public, private

SUPERCLASS

Declare the superclass of the class
If a superclass is not specified, it is assumed to be the object class. To explicitly specify the super class of a class, you must put the extends keyword plus the name of the superclass between the name of the class that has been created classClassName extends SuperClassName{
}.
A subclass inherits the variables and methods of the superclass

LIST THE INTERFACES IMPLEMENTED BY THE CLASS

When declaring a class you can specify what interface.
An interface declares a set of methods and constants without specifying their implementation for any method. When a class requires the implementation of an interface, it must provide the implementation for all methods declared in the interface.

To declare that a class implements an interface, the keyword implements must be entered followed by the list of interfaces implemented by the class, delimited by commas, for example an interface called Arithmetic that defines the methods called addition (), subtraction (), etc. The Imaginary Number class can declare that it implements the Arithmetic interface like this.class Imaginary Number extends Number implements Arithmetic.

DECLARE

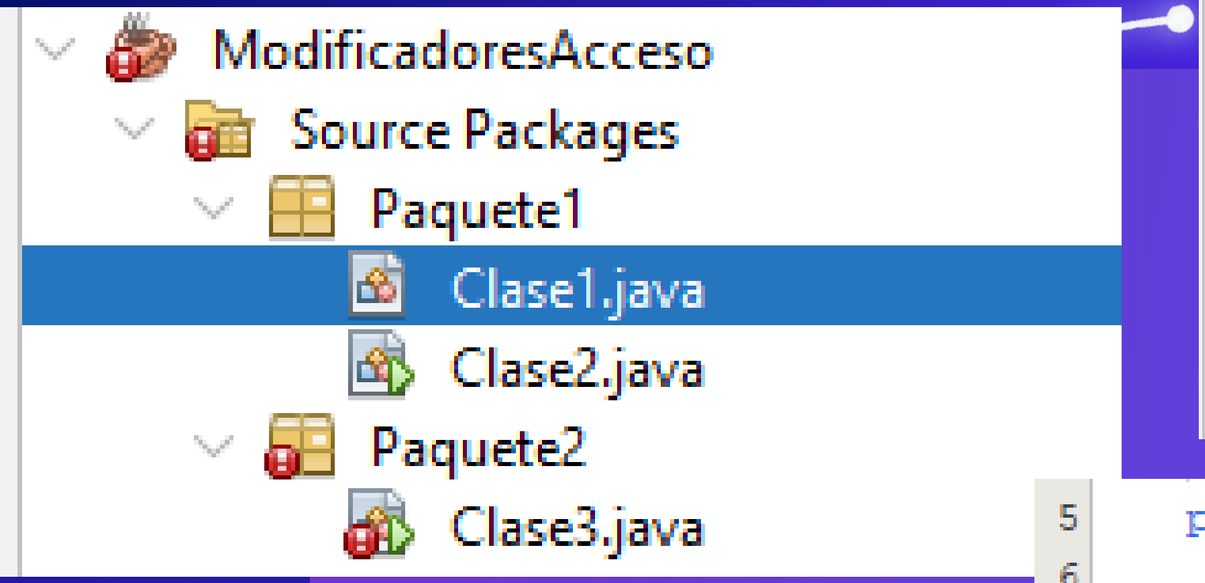
Declare if the class is public, abstract or final
The public modifier declares that the class can be used by objects outside the current package. When you use the word public you must ensure that it is the first item in the declaration. public class ImaginaryNumber extends Number implements Arithmetic {
}.

MODIFIER

- The abstract modifier declares that it is an abstract class, which is designed to be a superclass and cannot be instantiated
- The final modifier declares that a class is final, meaning that it cannot have subclasses. This is done for security reasons and design reasons. A class that contains unimplemented methods cannot be final.

ACCESS MODIFIER

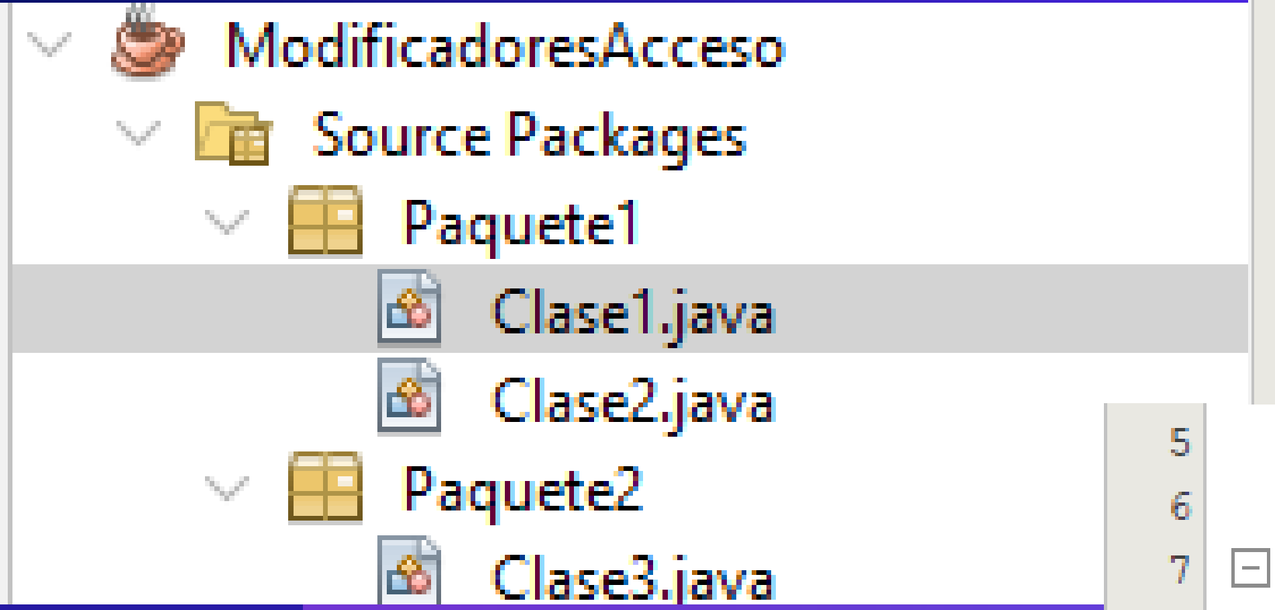
MODIFICADOR	CLASE	PACKAGE	SUBCLASE	TODOS
Public	Sí	Sí	Sí	Sí
Protected	Sí	Sí	Sí	No
Private	Sí	No	No	No
Por defecto	Sí	Sí	No	No




```
5   package Paquete1;
6
7   public class Clase2 {
8       public static void main(String[] args){
9           Clase1 obj1= new Clase1();
10
11          obj1.atributo1 = 15;
12      }
```

```
5   package Paquete1;
6
7   public class Clase1 {
8       int atributo1;
9
10
11  }
12
```

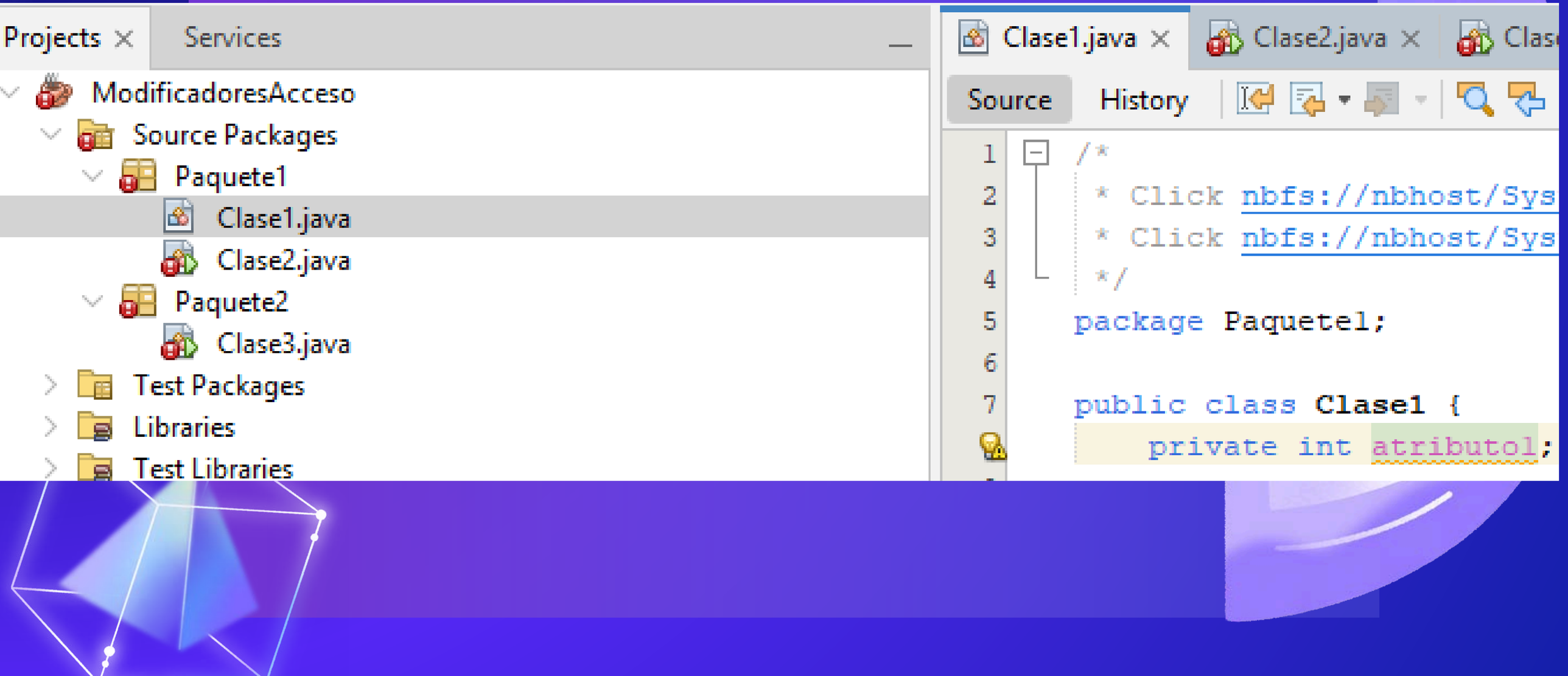
```
5   package Paquete2;
6
7   import Paquete1.Clase1;
8
9   public class Clase3 {
10      public static void main(String[] args){
11          Clase1 obj1 = new Clase1();
12          atributo1 is not public in Clase1; cannot be accessed from outside package
13          ----
14          (Alt-Enter shows hints)
            obj1.atributo1 = 15;
16
17      }
18
19  }
```

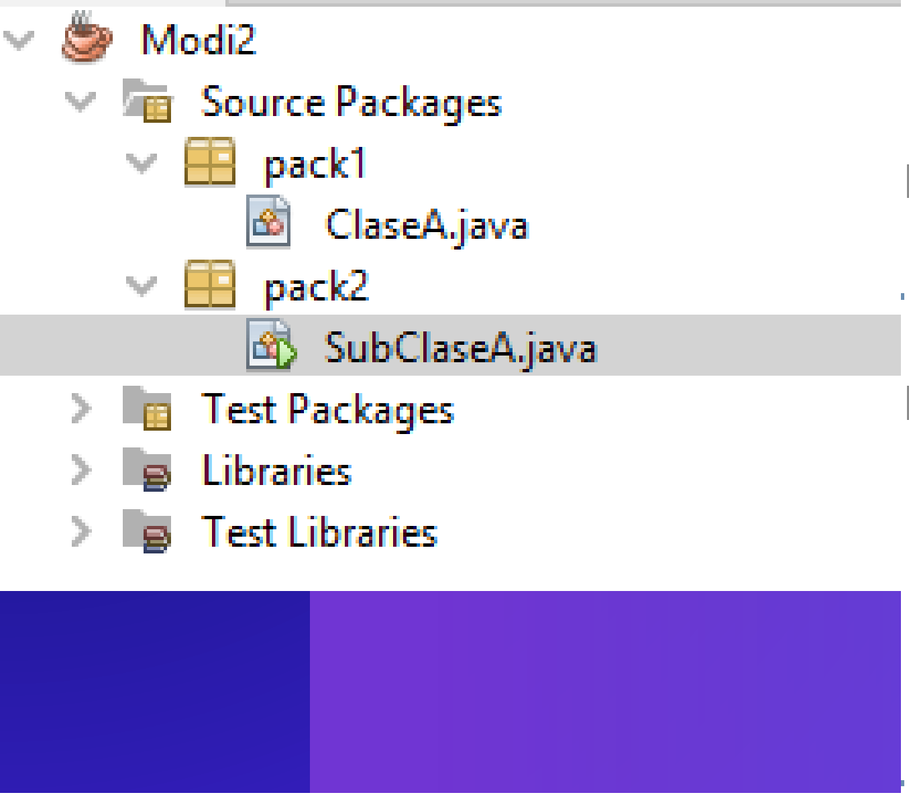



```java
package Paquete1;

public class Clase1 {
    public int atributo1;


}
```

```java
package Paquete1;

public class Clase2 {
    public static void main(String[] args){
        Clase1 obj1= new Clase1();

        obj1.atributo1 = 15;
```

```java
package Paquete2;

import Paquete1.Clase1;

public class Clase3 {
    public static void main(String[] args){
        Clase1 obj1 = new Clase1();



        obj1.atributo1 = 15;

    }

}
```


Projects
Services
ModificadoresAcceso
Source Packages
Paquete1
Clase1.java
Clase2.java
Paquete2
Clase3.java
Test Packages
Libraries
Test Libraries
Clase1.java
Clase2.java
Source
History
1 /*
2 * Click nbfs://nbhost/Sys
3 * Click nbfs://nbhost/Sys
4 */
5 package Paquete1;
6
7 public class Clase1 {
private int atributo1;



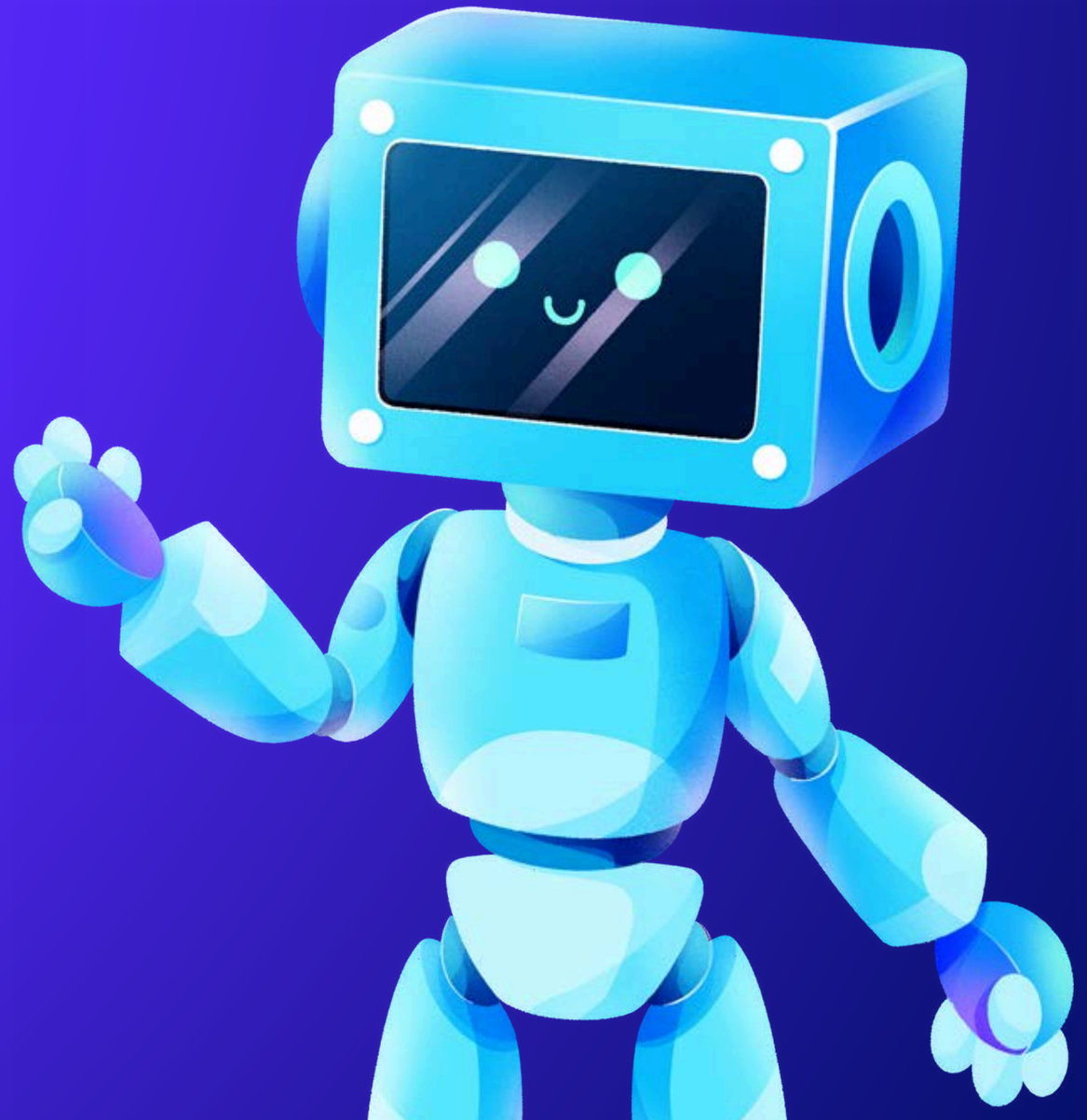

```java
package pack2;

import pack1.ClaseA; // Importar la clase del otro paquete

public class SubClaseA extends ClaseA { // Extiende ClaseA

    public static void main(String[] args) {
        // Creamos un objeto de SubClaseA
        SubClaseA asub = new SubClaseA();

        // Accedemos al atributo protegido de la clase base
        System.out.println( x: asub.MensajeProtegido);
    }
}
```

```java
package pack1;

public class ClaseA {
    protected String MensajeProtegido = "Esto está protegido";
}
```

Output - Modi2 (run)

```
run:
Esto está protegido
BUILD SUCCESSFUL (total time: 0 seconds)
```

THANKS



Notas

Primera exposición: 16

Segunda exposición: 20

Tercera exposición: 19