Programming

intermediate II

Object-oriented programming

A system is made up of several programs.

Each one responsible for some function.

Think of a bank, we have a function to create an account, request a credit card, among other functions.

You created a record for a customer to open their account.

Then the customer asks for a credit card and the age needs to be validated

(if you are over 18).

Moments later, it is necessary to validate the new age.

Next year, I wanted to make an investment and it needed to be confirmed again.

You remembered that you needed to change your age in all the old functions, because now it's older!

Can it get worse?

YES!

A new programmer has entered the company, can you remember all the places where the age needs to be validated?

Object-oriented programming, can help.

That being said, you will not have extended validation throughout the code.

If the client's age falls in a single place, it facilitates any exchange.

So, when there is any change, the rest of the functions will only receive this value, without having to worry.

Instances

INSTANCES

What every account has and is important to us?

- account number
- account holder name
- total value
- account type

INSTANCES

What every account does and is important to us?

That is, what would we like to "ask for on account"?

- withdraw an amount x from the account
- deposit an amount x
- print the name of the account holder
- return the current balance
- transfer a quantity x to another account y
- return the account type

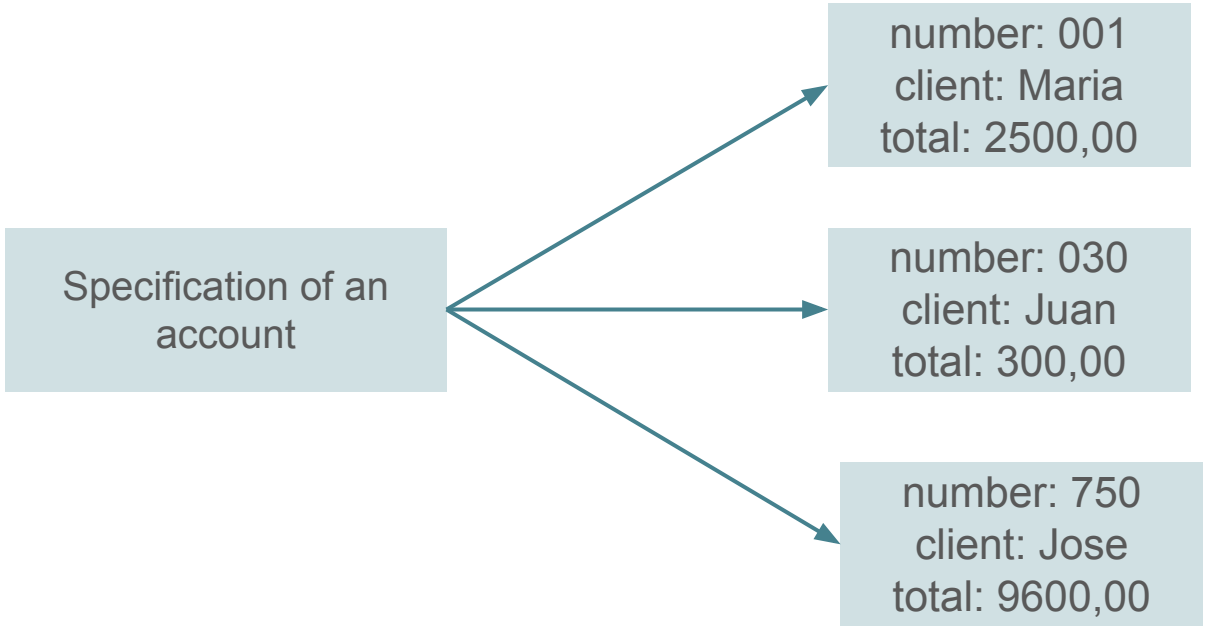
With that, we have the project of a bank account.

Can we take that project and access its balance? **NO**.

What we have is the project.

first we need to open an account to be able to access the features and ask the program to do something on the account.

INSTANCES



Look at the figure:

Although the paper on the left side specifies an account, is this specification an account?

Can we deposit and withdraw money from this image?

No, we use the account specification to be able to create instances that are really accounts, which we can perform the operations we created.

Despite declaring that every account has a balance, a number and an agency on paper (as on the left in the figure), it is in the instances of this project that there is actually space to store these values.

Like a house project. We can only use the bathroom in the house if we build it. This is called instances.
We instantiate the paper on which it was drawn and we will have a house.

Like a house project.
We can only use the bathroom in the house if we build it.
This is called **instances**.
We **instantiate** the paper on which it was drawn and we will have a house.

Classes and Objects

We call the project of the account, or of the house, **CLASS** .

Where we define, our initial design.

The paper where we put what the house should have.

What can we build, which would be a person's real account, or the real house, we call it **OBJECTS**

Another example: A pie recipe.

The question is: do you eat a cake recipe? **NO**!

We need to **instantiate** it, that is, create a cake object from that specification (the class) to use it.

We can create hundreds of cakes from that class (the recipe, in that case), they can be very similar, some even identical, but they are different objects.

Classes and objec

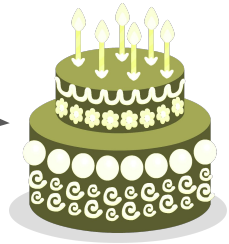
instantiate

Class recipe

string color
string candy_color

Object Pie 1

color: brown
candy_color : red



Object Pie 2

color: yellow
candy_color :yellow

Object Pie 3

color : pink
candy_color: pink

Attributes, Methods and Parameters

ATTRIBUTES

```
class account

    int number

    String holder_account

    double total_amount
```

Here we declare what every account must have.
This are ATTRIBUTES

Within a class, we also have to declare what each account does and how this is done.

For example, how does an Account withdraw money?

We specify this inside the Account class itself, and not somewhere outside.

Those are called **methods**. In other words, it is the way to do an operation with an object.

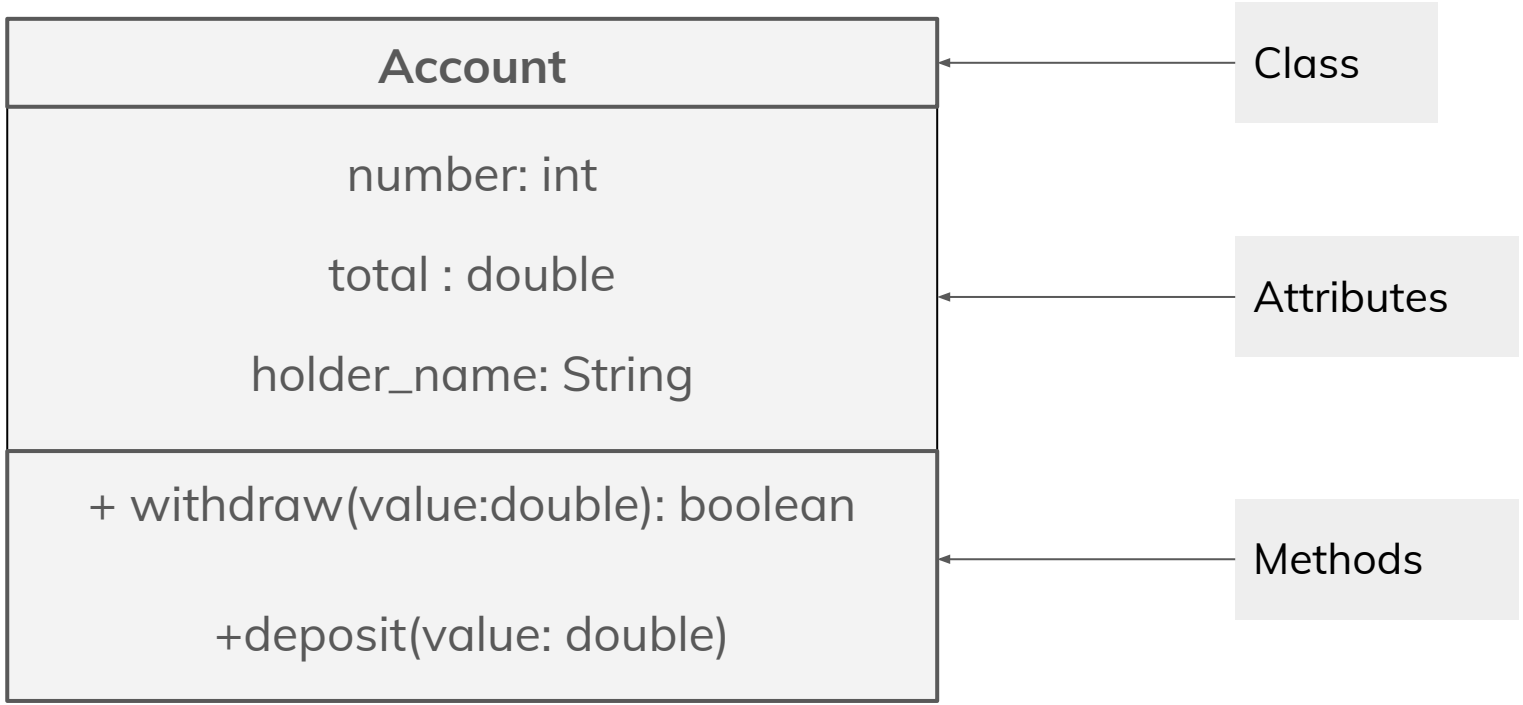
METHODS

```
method Withdraw (double amount_money_withdraw)

    double new_total = total - amount_money_withdraw

    total = new_total
```

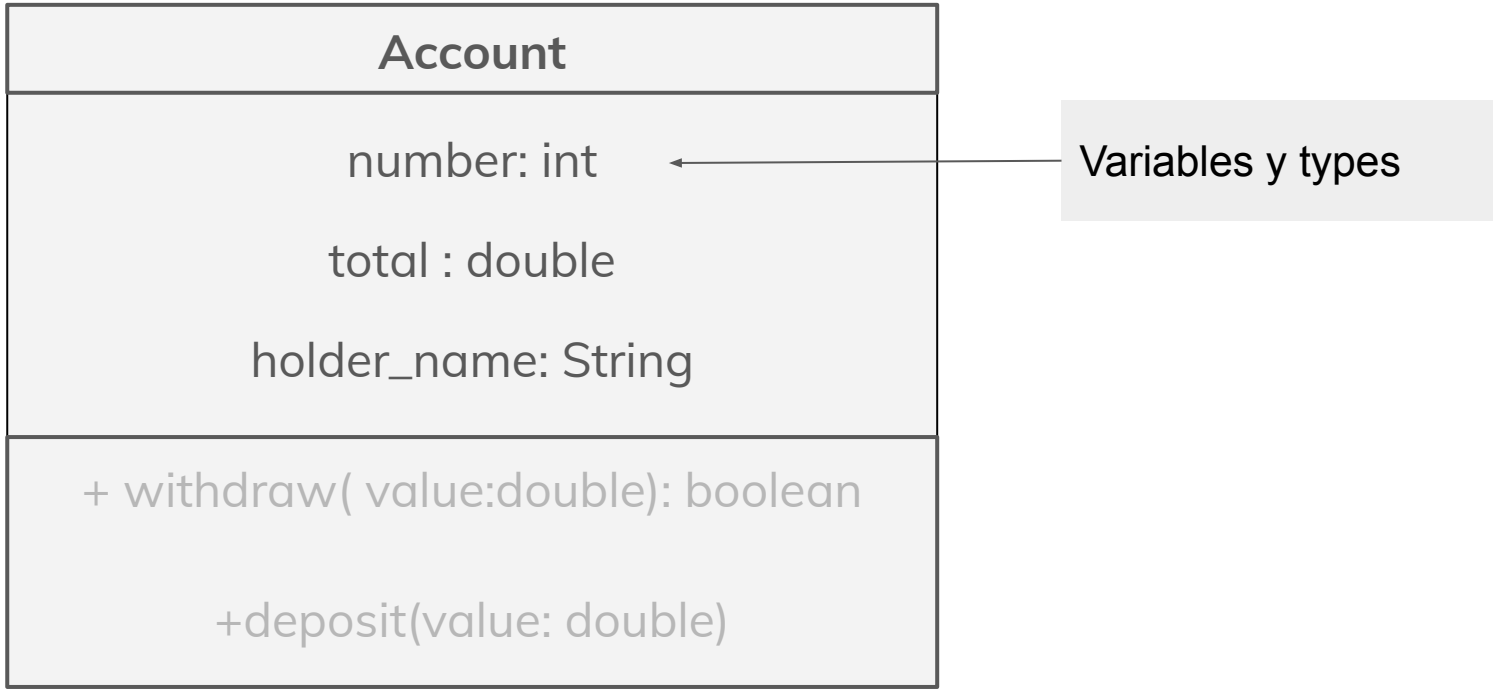
METHODS

```
method DepositMoney(double amount_money_deposit)
    double new_total = total - amount_money_deposit
    total = new_total
```

ATTRIBUTES AND METHODS



ATTRIBUTES


Account
number: int
total : double
holder_name: String
+ withdraw(value:double): boolean
+deposit(value: double)
Variables y types

METHODS

+deposit(value: double)

Method name → DEPOSIT

value we need as input: double

In other words, to make a deposit we need a value that is type double.

Example: We deposit a value of 2345.65

METHODS

+ withdraw(value:double): boolean

Method name → withdraw

value we need as input: double

method return: Boolean

In other words, to get money we need to know the value that is type double.

Ex: We get a value of 2345.65

In addition to that, we need to know if there is that money in the account. That is, a boolean, will return YES or NO.

METHODS

+ withdraw(value:double): boolean

```
method withdraw(double value)
    if value> total_amount_account
        print ("You do not have enough money")
    else
        total_amount_account= total_amount_account- value
        print ("Withdraw done")
```

METHODS

```
+ transfer_to( value:double, account_number: int): boolean
```

```
method transfer_to (double value, entero account_number)
    if value> total_amount
        print ("Enough money to transfer")
    else
        total_amount= total_amount- valor
        print ("Transfer done")
```

PARAMETERS

```
+ transfer_to ( value :double, account_number: int): boolean
```

PARAMETERS

They are values that the method needs.

In the example, to make a transfer, we need the value to know if you have enough money and the beneficiary's account, where the money is going.

Inheritance

Think of a bank, with an employee.

There is a manager, who is also an employee, but performs more functions than an employee.

Example:

The manager has more access to the system than an employee.

Do we need to create another class called manager?

Do we need to put all the methods again?

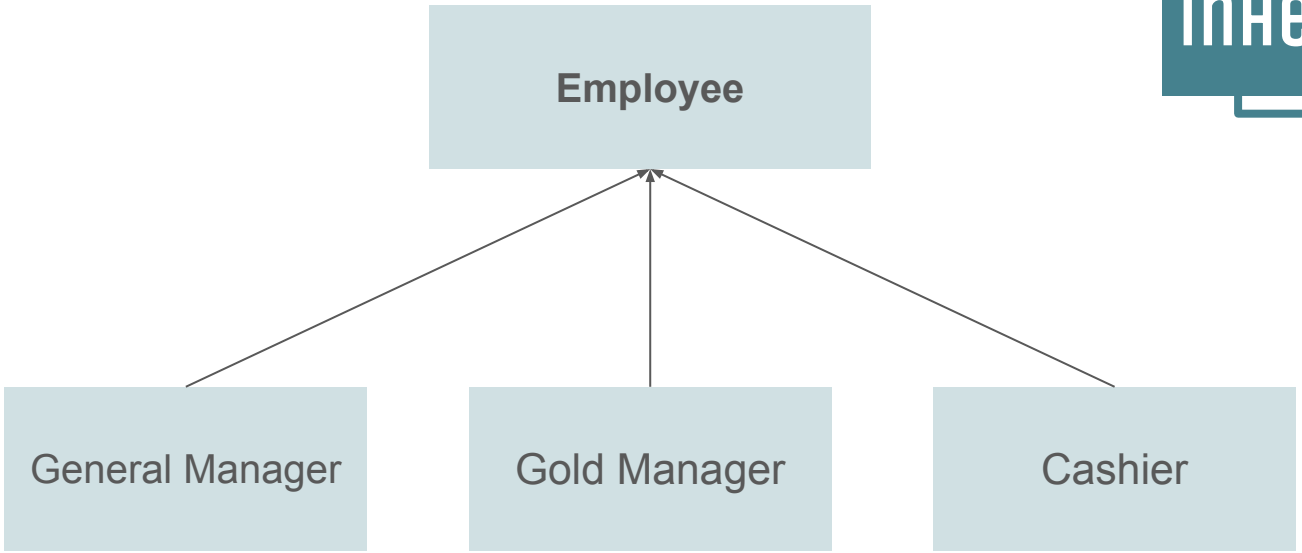
Can it get worse? yes

If the bank restructures the positions, and creates:

- **general manager of the bank, manager of the gold clients, and cashier of the bank.**

And all of them have several functions in common.

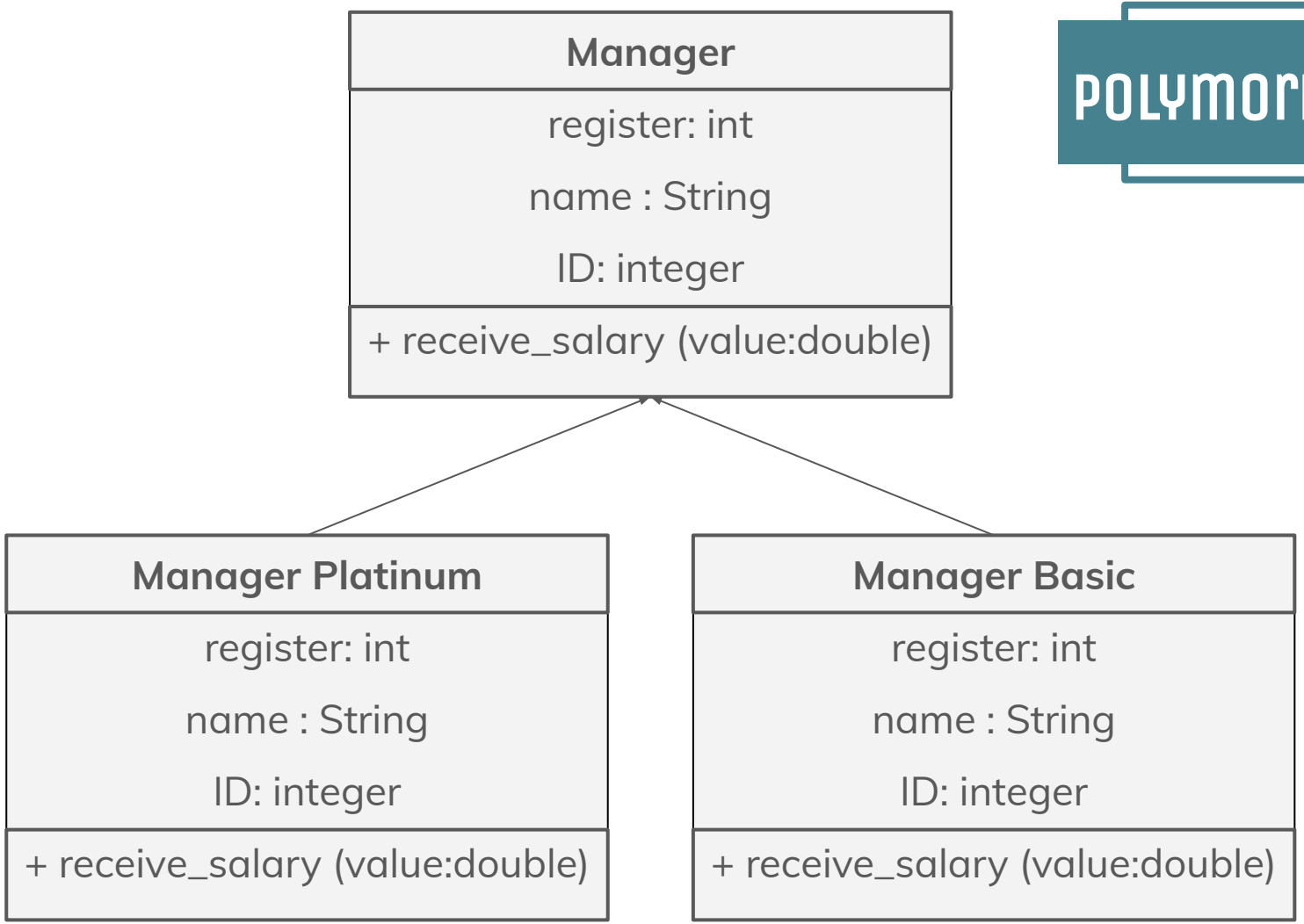
Inheritance



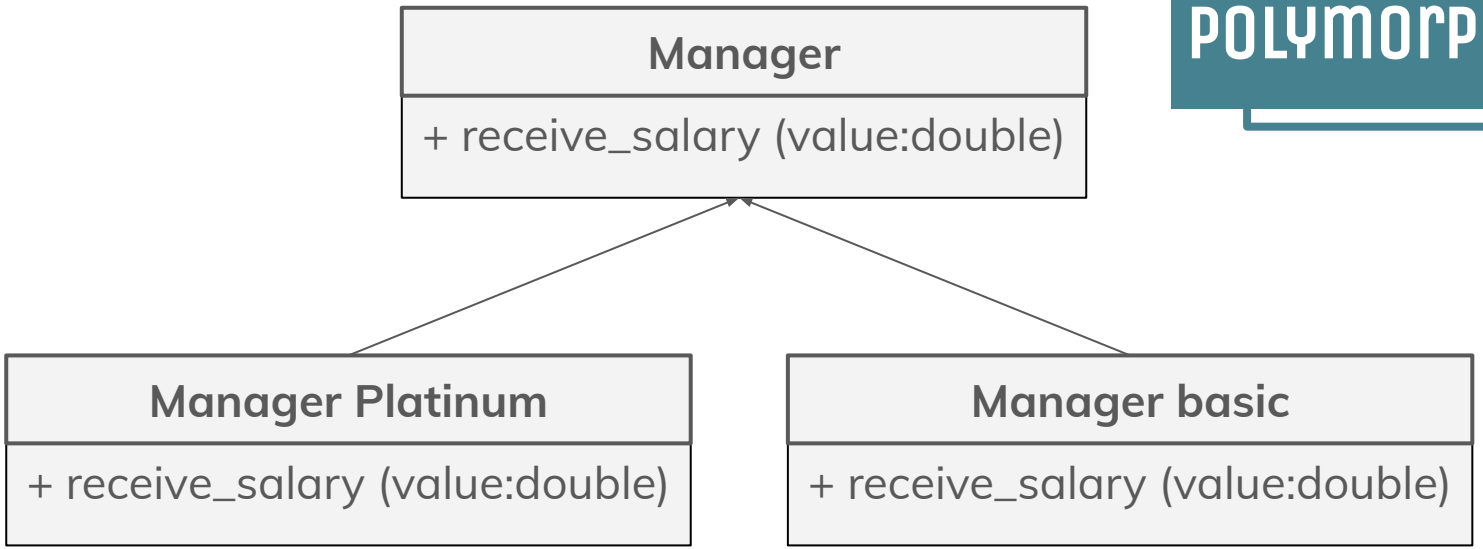
Look at the figure:
All positions are employees.
General Manager, Gold Manager and Cashier are employees.

Polymorphism

POLYMORPHISM



POLYMORPHISM



The Platinum manager receives **1%** on the number of accounts opened in the month. That is, if at the end of the month the agency has more than 100 open accounts, the manager earns 1% more in his salary

The Basic manager receives **2.5%** on the number of accounts opened in the month. That is, if at the end of the month the agency has more than 100 open accounts, the manager earns 2.5% more in his salary.

POLYMORPHISM

In other words, we have the same method, but with different behavior for each Object.

Each one of the managers, who receive the methods by inheritance, can make different calculations.

In the example one receives 1% and the other 2.5%

7

Learned and recommendations

WHAT WE LEARN

- Lessons
- Objects
- instances
- Inheritance
- Polymorphism

RECOMENDACIONES

- Abstract Class
- Interface
- Database
- Java Basic

THANKS