

# Apuntes Semana 2

---

## Clase 1

*Presentacion de la materia Programación II.* 😊

- Presentación del profesor.
- Presentación de los estudiantes.
- Información de varios recurso que se necesitaran para los talleres.
- El primer taller en grupo se trataba de armar una torre lo mas alta y con la menor cantidad de material posible (spoiler: no salio muy bien que se diga :p).
- Explicación de como iban a ser la evaluación.
- Las pruebas y la duración de las mismas, las pruebas serian de dos horas, en cambio los examenes de ocho horas.
- El proyecto, se deberia de entregar la primera parte en el primer bimestre y la finalizacion del mismo en el segundo bimestre.
- Cuadro de evaluaciones y sus porcentajes:

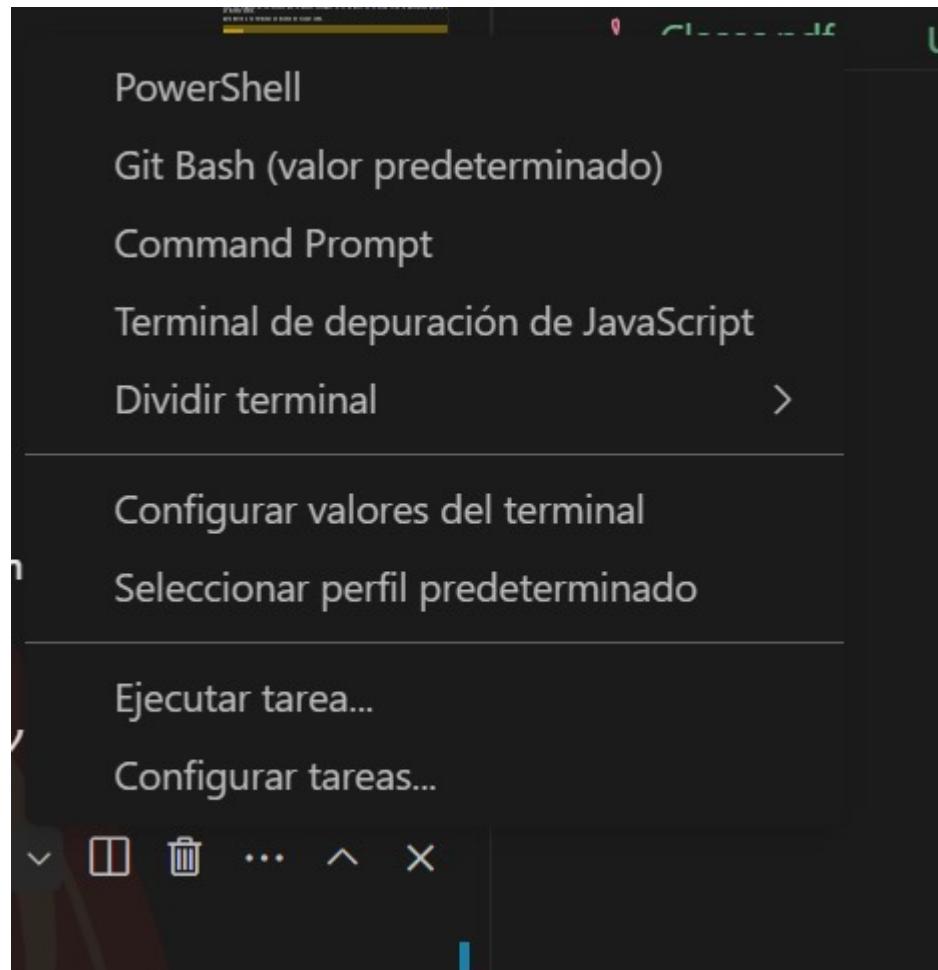
Evaluación	Puntaje	Bonos Extras	Puntaje
-Prueba	25%	-Actuación	+0.1 punto
-Examen	25%	-Retos	+1.0 punto
-Workshop	10%		
-Homework	10%		
-Proyecto	30%		

- Explicación de como se iba a llevar el curso, las aplicaciones que se deberan de instalar y como conseguir una buena optimización, ademas de la personalizacion de las mismas, haciendo que sean mas comodas para sacarles el maximo provecho al momento de trabajar y tener buenos resultados.
- Presentar los apuntes de markdonw al final de cada bimestre.
- Talleres en grupo.
- Proyecto en grupo, el proyecto debe tener un aparato externo.
- Presentación de la normativa.
- Personalizacion de las aplicaciones de trabajo.
- Aprender los comandos.

## Clase 2

### Personalización del Visual Code y de Git bash

- Para la personalización del Git Bash y del visual code, se puede personalizar su terminal para que esta sea la determinada en Visual Code y salga por defecto automáticamente, cada vez que se le llame con el comando CTRL + Ñ o manualmente.



Ademas se puede personalizar el git bash en si con la ayuda de otras aplicaciones externas, como por ejemplo: *Oh my posh* que es una de varias que pueden ayudar a darle un toque mas personal a la terminal y la consola de Git bash, con varios diseños y fuentes de se pueden instalar facilmente.

```
$ exec bash
Home ~ ✓ nano .bashrc
Home ~ ✓ exec bash
Celeste ~ 0ms 4:02 PM
Home >> nano .bashrc
Celeste 36.749s 4:03 PM
Home >> exec bash
[ Home@Celeste 0ms
$ nano .bashrc
[ Home@Celeste 20.864s
$ exec bash

37 RAM: 11/31GB 0ms
bash 4:05:43 PM | Saturday nano .bashrc

37 RAM: 11/31GB 16s 267ms
bash 4:06:38 PM | Saturday exec bash
~ Oms Home bash

> nano .bashrc
~ 14.703s Home bash

> exec bash
Home ~ 16:07 exec bash
Home ~ 16:07 nano .bashrc
Home ~ 16:08 exec bash
Home ~ 16:08 nano .bashrc
Home ~ 16:09 exec bash
Home@Celeste ~ nano .bashrc
Home@Celeste ~ exec bash
Home ~ nano .bashrc
Home ~ exec bash
Home ~ 16:12

16:11:14 0ms
```

Esos son algunos de los diseños que se pueden conseguir en Oh my posh, de la misma forma la aplicación permite crear un diseño nuevo, al final de la personalización se puede obtener un resultado así en la terminal.

```

[...] Celeste > touch nota1.java
[...] Celeste > git config user.name
[...] Celeste > git config user.email
[...] Celeste > git config user.name
[...] Celeste >

```

- De la misma forma se puede personalizar el color de visual estudio con ayuda de los temas que vienen en la aplicación, pero tambien se pueden instalar otros, así mismo con los iconos de las aplicaciones o archivos que se abran dentro del visual.
- Tambien es recomendable cambiar la letra del visual code y de git bash para que sea mas comoda al momento de trabajar, pero eligiendo una que sea entendible y que facilite el trabajo.

Con todo eso listo, se puede tener una buena herramienta, que sea comoda y al gusto del usuario para trabajar adecuadamente.

### Comandos 😊

- **Windows**

- CTRL + SHIFT + P: para abrir la barra de comandos donde estan las aplicaciones y funciones.
- CTRL + P: para ver los documentos que se han abierto recientemente o buscar algun archivo para abrirlo.
- CTRL + B: para aparecer y desaparecer la barra donde se observan las carpetas y documentos que estaban abiertos o almacenados.
- CTRL + D: para seleccionar varias lineas.
- CTRL + S: para guardar.
- CTRL + Ñ: para aparecer y desaparecer la terminal.
- SHIFT + ALT + UP/DOWN: para copiar y pegar una linea del codigo.
- ALT + ->: Para moverse entre los archivos que se encuentran abiertos en la barra de arriba.
- ALT + ->(arriba): Para mover una linea del codigo ya sea arriba o abajo.
- SHIFT + ALT + A: para poner una linea de comentario.
- SHIFT + ALT: para seleccionar la linea o palabra.

- **Linux**

- pwd: para saber en que carpeta actual te encuentras.
- touch: para crear un nuevo archivo de cualquier tipo.
- code: para abrir el documento creado.
- ls: lista de archivos en el directorio.
- clear: limpiar consola.

## Clase 3

## Uso \*del Markdown

Markdown es una herramienta para tomar apuntes sin necesidad de una herramienta fuera del área de trabajo, visual code, este se puede descargar como un archivo en formato PDF para compartir. Para eso se necesitan algunas extensiones.

- Título: #Título.

Mientras mas # tenga se irá dividiendo en títulos y subtítulos.

- Cursiva: \*Cursiva\*

Con solo un \* a cada lado de la palabra u oración.

- Negritas: \*\*Negritas\*\*

Con dos \*\* a cada lado de la palabra u oración.

- Negrita y Cursiva: \*\*\*Negrita y Cursiva\*\*\*

Con tres \*\*\* a cada lado de la palabra u oración.

- Link de página o sitio web: [Nombre del link](Enlace del link)

## Youtube

- Imagen: ![Nombre de la imagen](Dirección o link de la imagen)

```

[...] Celeste > \OneDrive\SAKI 106ms 6:00 PM
  Home >> touch nota1.java
[...] Celeste > \OneDrive\SAKI 105ms 6:03 PM
  Home >> git config user.name
Celeste040205
[...] Celeste > \OneDrive\SAKI → ( main) 115ms 6:24 PM
  Home >> git config user.email
inu-kun_2005@hotmail.com
[...] Celeste > \OneDrive\SAKI → ( main) 124ms 6:24 PM
  Home >>

```

- Código: ``` Nombre del lenguaje de programación

(y cierra en una siguiente línea) ```

Tres ` para formar el cuadro y el nombre del lenguaje para que así se resalte el código de acuerdo al mismo.

```

#include <iostream>
#include <cstdlib>
using namespace std;
struct nodo{
    int nro;
}

```

```
    struct nodo *izq, *der;
};
```

- Cuadros: se puede formar cuadros dividiendo las columnas con | y la primera fila (titulos del cuadro) con ---

<b>titulo 1</b>	<b>titulo 2</b>
ejemplo 1	ejemplo 2
ejemplo 1	ejemplo 2

- Comando:
  - CTRL + SHIFT + V: para ver la vista previa de markdown.

### Manejo de git/github

- **GIT**

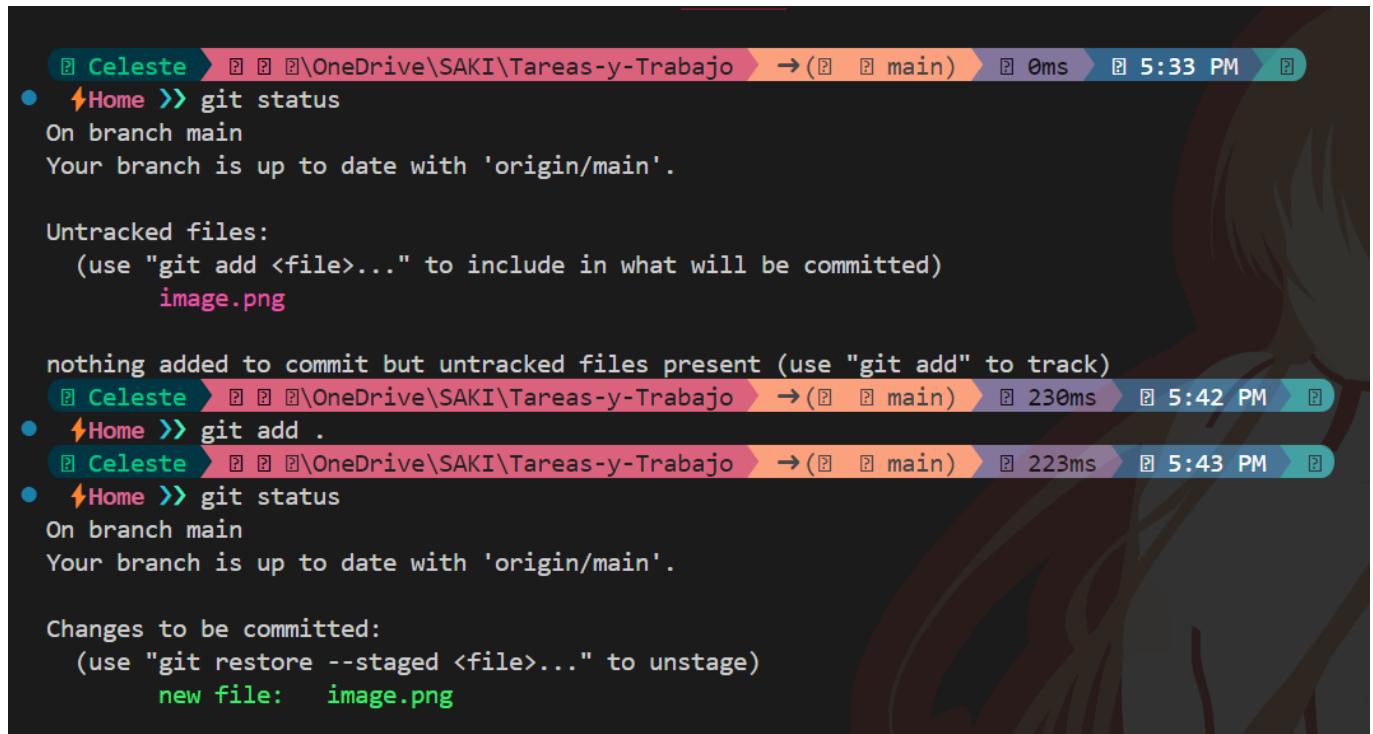
- git init: crear un nuevo repositorio.
  - git status: ver como se encuentran los archivos creados, si estan dentro de git o no.
  - git add . : añadir todos los archivos de la lista que se presente tras el *git status*.
  - git push: para subir a la nube.
  - git pull: para bajar de la nube los cambios efectuados.
  - git clone: para clorar un repositorio.
  - git commit: para guardar en la nube.
- Subir los archivos a la nube.
  - Crear un repositorio en GitHub.
  - Crear una carpeta en archivos en la cual clonar el repositorio del Github.

```
/usr/bin/bash --login -i
[celeste] ~ % \OneDrive\SAKI % main 0ms 5:24 PM
$ Home >> git clone https://github.com/Celeste040205/Tareas-y-Trabajo.git
fatal: destination path 'Tareas-y-Trabajo' already exists and is not an empty directory.

[celeste] ~ % \OneDrive\SAKI % main 147ms 5:25 PM
$ Home >> git clone https://github.com/Celeste040205/Tareas-y-Trabajo.git
Cloning into 'Tareas-y-Trabajo'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (6/6), 274.79 KiB | 538.00 KiB/s, done.

[celeste] ~ % \OneDrive\SAKI % main 2.065s 5:26 PM
$ Home >>
```

- Entonces se clonara en una nueva carpeta, con el nombre del repositorio.
- En esa carpeta se podra almacenar los documentos para subir en la nube.
- Una vez efectudos los cambios y demas, poner `git status` para comprobar como estan los cambios y `git add .` para agregar todos los cambios, comprobar con `git status` para ver todo en orden.



```

❯ Celeste ➜ 🌐 OneDrive\SAKI\Tareas-y-Trabajo ➔ (main) 0ms 5:33 PM
● ⚡Home >> git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    image.png

nothing added to commit but untracked files present (use "git add" to track)
❯ Celeste ➜ 🌐 OneDrive\SAKI\Tareas-y-Trabajo ➔ (main) 230ms 5:42 PM
● ⚡Home >> git add .
❯ Celeste ➜ 🌐 OneDrive\SAKI\Tareas-y-Trabajo ➔ (main) 223ms 5:43 PM
● ⚡Home >> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   image.png

```

- y entonces con el comando de `git commit` insertar, los cambios.

### *Resumen del trabajo*

Clonar y subir cambios a la nube Crean una carpeta en la cual se va a clonar el proyecto y dan click derecho "git bash here"

- GitBash
  - `git clone "link del repositorio"`.
- VsCode
  - Ctrl + N: abrir terminal
  - `git status`
  - `git add`
  - `git commit -m "cualquier nombre random" (cambios)`
  - `git remote -v: verificar si está sincronizada con el repositorio`
  - `git push origin main: subir el archivo.`
  - `git log: comprobar.`

## Apuntes Semana 3

---

### Clase 4

## *Programacion orientada a objetos*

- Java
  - Nace por problemas con el compilador, por las diferencias entre un sistema y otro.
  - Ayuda a la distribucion del los codigos complementando los codigos del compilador.
- Funcionamiento:
  - codigo
  - compilacion
  - bytecode
  - JVM
  - Multiplataforma

## *Tipos de lenguaje*

- Compilador:

No ejecuta el programa si tiene el minimo error, proceso de compilacion que se presenta en el "trabajo" solo se tiene consola no el F5.
- Interprete:

Es mas permisivo y lee el codigo, se detiene al encontrar un error.



*Tipos de Java*

Se tienen varios.



### Estructura del lenguaje

- *package*: conjunto de librerias.
- *import*: escoger una libreria.
- *\*\*\**: para traer, pero es mejor traer de forma mas especifica para no cargar demaciado el sistema.
- *class*: usa clases para ejecutar el codigo, debe estar en minusculas.
- *string*: tipos
- *public static void main*: aprender debe estar alli.

### Ejemplo

```

1 package team.ed.course;
2 import java.lang.*;
3 public class Person {
4     private String name;
5     public static void main(String args[]){
6         Person friend = new Person();
7         friend.name = "Peter";
8         System.out.println("Hola " +
9             friend.name);
10    }
11 }

```



### Diferencias

Existen varias diferencias entre las estructuras y Programación orientada a objetos (O.O) de como funcionaba en C++ y como funciona ahora en Java.

Estructura	O.O
include (libreria)	import (paquetes)
funciones	metodos
struct	clases
variables	propiedades

- Propiedades:
  - Ambito: public(+), private(-), protec, friendly. (si se olvida de colocar esto el programa solo se pone en privado, en pocos casos en publico)
    - en URL tienen simbolos.
  - Tipo de dato: int, char.
  - Nombre
- Metodos: Las variables si existen solo que se encuentran dentro de los métodos, no se puede tener fuera de estos *si se tiene fuera se convierten en propiedad.*

- Método que retorne valor.
- Método sin retorno (void).

*Diferencia entre punteros En almacenamiento es lo mismo, pero al referirse en C es mas complicado que en Java, en c se puede darña facilmente y se debe gestionar "manualmente" los punteros, mientras que en Java esto se realiza de forma automatica, ya no es necesario todo el procedimiento que se da en C, basicamente Java facilita el proceso.*

Se pueden utilizar diagramas de flujo y el URL.

NO OLVIDAR EL PUNTO Y COMA (😉 :v

### Aplicaciones

Java gana en aplicaciones industriales.

### Diagrama de flujo

Es importante la ilustración de digramas de flujo para ver de forma grafica el código.

### Diagrama de flujo

Símbolo	Nombre	Función
	Inicio / Final	Representa el inicio y el final de un proceso
	Línea de Flujo	Indica el orden de la ejecución de las operaciones. La flecha indica la siguiente instrucción.
	Entrada / Salida	Representa la lectura de datos en la entrada y la impresión de datos en la salida
	Proceso	Representa cualquier tipo de operación
	Decisión	Nos permite analizar una situación, con base en los valores verdadero y falso

**TRACE**

Pseudocódigo. Pruebas de escritorio, seguimiento a la corrida de un programa. Automatizar el TRACE.

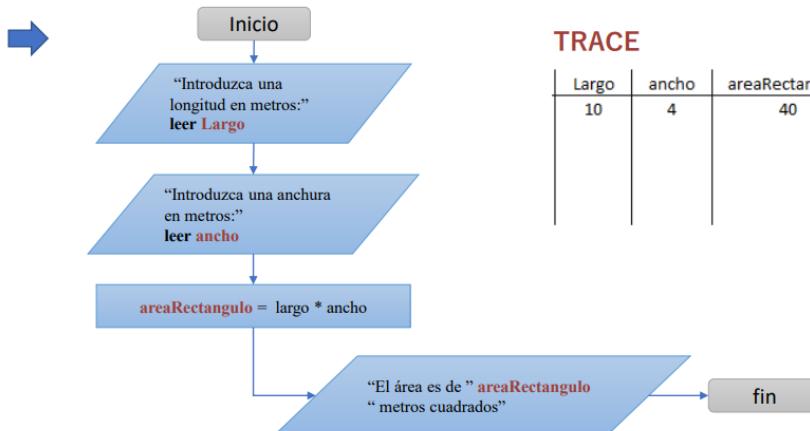
- Errores: el como esta escrito, mayusculas, minusculas, etc. Todo eso afecta.

- Forma correcta:

```

imprimir "Introduzca una longitud en metros:"
leer longitud
imprimir "Introduzca una anchura en metros:"
leer ancho
asignar a areaRectangulo = largo * ancho
imprimir "El área es de" areaRectangulo "metros cuadrados"

```

**TRACE**

Largo	ancho	areaRectangulo	Salida -> Terminal
10	4	40	Introduzca una longitud en metros 10 Introduzca una anchura en metros 4 El área es de 40 metros cuadrados

**ISSUE <> ERROR <> BUG**

- For incorrecta:

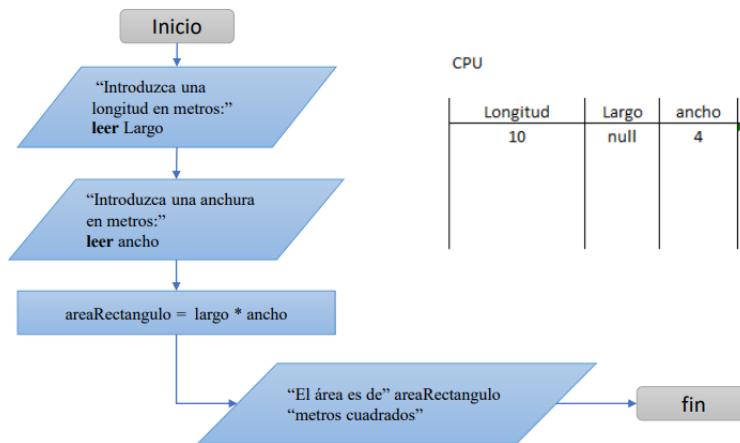
```

imprimir "Introduzca una longitud en metros:"
leer longitud
imprimir "Introduzca una anchura en metros:"
leer ancho
asignar a areaRectangulo = largo * ancho X
imprimir "El área es de" areaRectangulo "metros cuadrados"

```

**INTERPRETE <> COPILADOR**

.exe  
.class

**CPU**

Longitud	Largo	ancho	areaRectangulo	Salida -> Terminal
10	null	4	#VALUE!	Introduzca una longitud en metros 10 Introduzca una anchura en metros 4 El área es de <null> metros cuadrados

**Primer Código**

SHIFT + CNTRL + P

Poner un entorno base, en blanco, nuevo.

- pide que guardes en una carpeta.

Nombre: Proyecto 1.

- src: código fuente.

```
public class Hi {
    //propiedades aquí afuera.
    public static void main(String[] args){
        //métodos, funciones.
        int n = 10;
        for( int i = 0; i < n; i++){
            System.out.println(i);
        }
    }
}
```

- Compilar y Ejecutar desde consola: pwd: donde se encuentra. compilar: javac *Nombre del archivo*  
Ejecutar: *Nombre del archivo*

COMO FUNCIONA UN *for*

```
int n = 10;
for( int i = 0; i < n; i++)
    System.out.println(i);
```

(int i = 0): se puede poner fuera, se pone dentro para ahorrar una linea. (i < n): se puede cambiar a true or false TRUE: acepta el código, condición. FALSE: NO FUNCIONA, es como decir que no es real o falso lo que dice, entonces el código dice: "para que pones si es falso? :v". (i++/i--): para que sume o reste.

*for para que comience en 2 y se aumente de 2 en 2*

```
int n = 10;
for( int i = 2; i < n; i+=2)
    System.out.println(i);
```

(i+=2): forma abreviada para aumentar ya no solo de 1 en 1, sino más. (i=i+2): forma no abreviada.

CONDICIONAL (*if*)

```
int n = 10;
for( int i = 0; i < n; i++)
    if (i == 6)
        System.out.println("hay un seis");
    else
        System.out.println(i);
```

cuando toque en el número 6 en la ejecución, en lugar de salir "6" saldrá "hay un seis".

(else): es parte de la misma instrucción del if.

```
public class Hi {  
    //propiedades aqui afuera.  
    public static void main(String[] args){  
        //metodos, funciones.  
        int n = 10;  
        for( int i = 2; i < n; i+=2){  
            if (i == 6)  
                System.out.println("hay un seis");  
            else  
                System.out.println(i);  
        }  
    }  
}
```

codigo mas compacto con **OPERADOR TERNARIO**

```
public class Hi {  
    //propiedades aqui afuera.  
    public static void main(String[] args){  
        //metodos, funciones.  
        int n = 10;  
        for( int i = 2; i < n; i+=2){  
            System.out.println("hay un seis");  
        }  
    }  
}
```

(terminar de poner el codigo compacto xd)

## Clase 5

*Bucles*

*For*

Da un inicio y un final.

- creacion de documentos
- pretty format (instalar) no se ve donde se cierra los corchetes instalar extension.
- instanciacion: bf = New Buclefor() (crear o dar vida xd, )
- ejecucion power mod

```
public class App {  
    Run | Debug  
    public static void main(String[] args) throws Exception {  
        int a;  
        Buclefor buclefor;  
  
        //instancias new  
        buclefor = new Buclefor();  
        buclefor.SignoAlterno();  
    }  
}
```

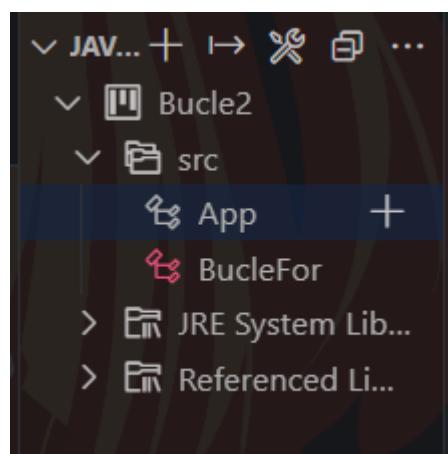
Revison de errores que pueden suceder en la plataforma para no compile.

## Clase 6

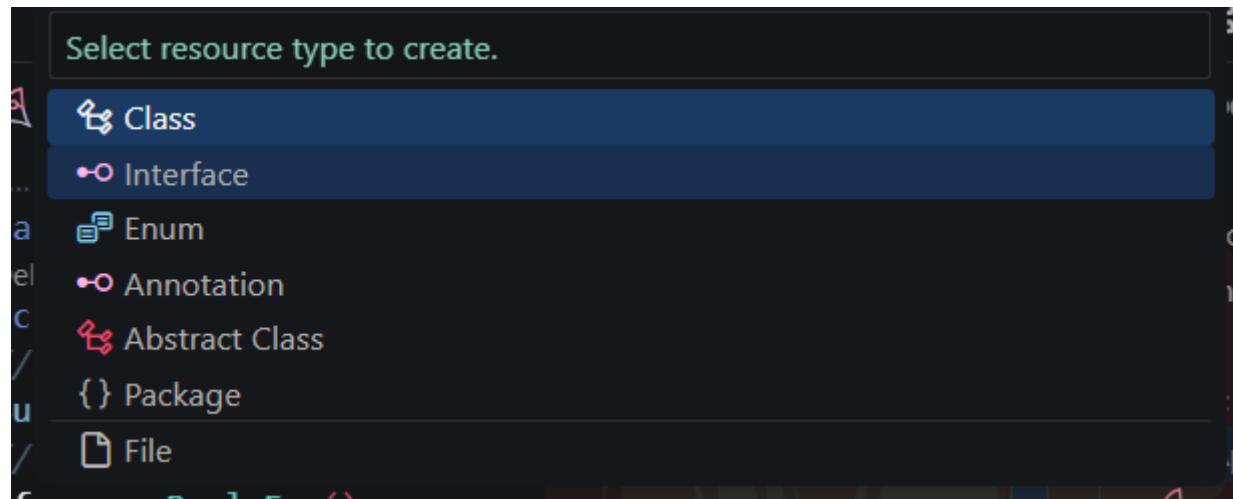
Bucles for parte 2

- Orden del directorio.
- Proyectos pequeños en distintas carpetas.
- Crear un numero proyecto.
- primero se debe saber cual sera el resultado del codigo para poder trabajar.

al momento de crear un nuevo proyecto se abrirá una nueva pestaña de Java donde está guardado y se tiene un "src" una fuente y un *App* que ayudará a la compilación.



A partir de esta carpeta se puede abrir un nuevo archivo que puede ser de varios tipos.



En este ocasión se selecciono *Class* y se le coloco el nombre *BucleFor* (las mayusculas son importantes) se comenzó a trabajar en este

A screenshot of the Java IDE showing two code editors. The left editor is for 'App.java' and contains the following code:

```
1 public class App {  
2     public static void main(String[] args) throws Exception {  
3         BucleFor;  
4     }  
5 }
```

The right editor is for 'BucleFor.java' and contains the following code:

```
1 public class BucleFor {  
2     // método : + - + - + - + -  
3     public void signoAlternado(){  
4         for (int i = 1; i < 10; i++) {  
5             if(i%2==0)  
6                 System.out.println("-");  
7             else  
8                 System.out.println("+");  
9         }  
10    }  
11 }  
12 }  
13 }
```

### Código de For compilado

The screenshot shows a Java development environment with two files open:

- App.java**:

```
1  public class App {  
2      Run | Debug  
3      public static void main(String[] args) throws Exception {  
4          //declarar  
5          BucleFor bf;  
6          //instanciar  
7          bf = new BucleFor();  
8          //llamar el metodo  
9          bf.signoAlterno();  
10     }  
11 }
```
- BucleFor.java**: (Partially visible)

Below the editor, there are several tabs: PROBLEMAS, SALIDA, CONSOLA DE DEPURACIÓN, TERMINAL, and others. The TERMINAL tab is active, showing the output of running the application:

```
Celeste ➤ ② ③ ④ \OneDrive\SAKI\progra\projec2\Bucle2 ➔ (② ③ main) ➔  
④  
⚡Home ➤ /usr/bin/env C:\\Program\\ Files\\Java\\jre-1.8\\bin\\java.exe  
\\OneDrive\\SAKI\\progra\\projec2\\Bucle2\\bin App  
● FOR:  
+ - + - + - + - +
```

Error (Depuración)



- Niveles (variables): for (n)
  - un palito, se aumenta, cierra la escalera.
    - nivel 1: |\_
    - nivel 2: |\_
    - nivel 3: |\_
  - se aumentan n niveles.
- aumentar la escalera a lo ancho (otra variable): for (h)
  - aumentar un espacio entre cada uno.

```

App.java
src > App > App > main(String[])
You, hace 4 minutos | 1 author (You)
1 public class App {
2     Run | Debug
3     public static void main(String[] args) throws Exception {
4         //declarar
5         BucleFor bf;
6         //instanciar
7         bf = new BucleFor();
8         //llamar el metodo
9         bf.signoAlterno();
10        bf.signoAlternoGenerativo();
11        bf.signoAlternoGenerativoSecond();
12        bf.escalera();
13    }
14 }

BucleFor.java
src > BucleFor > BucleFor > escalera()
1 public class BucleFor {
2     public void signoAlternoGenerativoSecond(){
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
}
}

//metodo: escalera.
public void escalera(){
    System.out.println("Escalera:");
    int nivel = 10;
    String sacalon= "|";
    for (int i = 1; i < nivel; i++){
        for (int e = 1; e < i; e++){
            System.out.print(" ");
        }
        System.out.println(sacalon);
    }
}

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS GITLENS

Escalera:

```

|_
|_|_
|_|_|_
|_|_|_|_
|_|_|_|_|_
|_|_|_|_|_|_
|_|_|_|_|_|_|_
|_|_|_|_|_|_|_|_
|_|_|_|_|_|_|_|_|_
|_|_|_|_|_|_|_|_|_|_
|_|_|_|_|_|_|_|_|_|_|_

```

*While*

pasar de un for a while.



The screenshot shows two Java code editors in a dark-themed IDE. The left editor contains `BucleWhile.java` with the following code:

```

src > BucleWhile.java > ...
1 public class BucleWhile {
2     //metodo: + - + - + - +
3     public void signoAlterno(){
4         System.out.println("While:");
5         int i = 1;
6         while (i < 10) {
7             if (i%2==0)
8                 System.out.print("- ");
9             else
10                System.out.print("+ ");
11            i++;
12        }
13        System.out.println();
14    }
15 }
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35

```

The right editor contains `App.java` with the following code:

```

src > App.java > App > main(String[])
You, hace 32 segundos | 1 author (You)
1 public class App {
2     Run | Debug
3     public static void main(String[] args) throws Exception {
4         //declarar
5         BucleFor bf;
6         BucleWhile bw;
7
8         //instanciar
9         bf = new BucleFor();
10        bw = new BucleWhile();
11
12         //Llamar el metodo
13         bf.signoAlterno();
14         bf.signoAlternoGenerativo();
15         bf.signoAlternoGenerativoSecond();
16         bf.escalera();
17
18         bw.signoAlterno();
19     }
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35

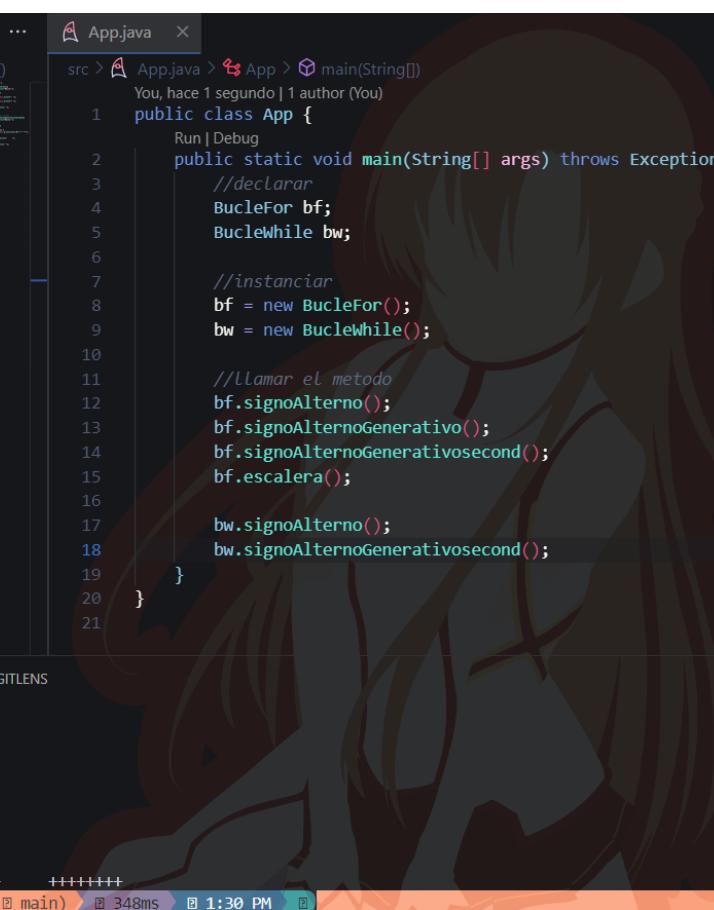
```

Below the editors are tabs for PROBLEMAS, SALIDA, CONSOLA DE DEPURACIÓN, TERMINAL, PUERTOS, and GITLENS. The TERMINAL tab is selected, showing the output of the program:

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS GITLENS
While:
+ - + - + - + - +

```



The screenshot shows three Java code editors in a dark-themed IDE. The left editor contains `BucleWhile.java` with the following code:

```

src > BucleWhile.java > BucleWhile > signoAlternoGenerativoSecond()
1 public class BucleWhile {
2     //metodo + - + + - - + + + + -
3     public void signoAlternoGenerativoSecond(){
4         System.out.println("While:");
5
6         int i = 1;
7         while (i < 10) {
8             int s = 1;
9             while (s < i) {
10                 System.out.print((i%2==0)? "- ":"+");
11                 s++;
12             }
13             System.out.print("      ");
14             i++;
15         }
16         System.out.println(" ");
17     }
18 }
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35

```

The middle editor contains `BucleFor.java` with the following code:

```

src > BucleFor.java > BucleFor > signoAlternoGenerativoSecond()
1 public class BucleFor {
2     //metodo + - + + - - + + + + -
3     public void signoAlternoGenerativoSecond(){
4         System.out.println("While:");
5
6         int i = 1;
7         while (i < 10) {
8             int s = 1;
9             while (s < i) {
10                 System.out.print((i%2==0)? "- ":"+");
11                 s++;
12             }
13             System.out.print("      ");
14             i++;
15         }
16         System.out.println(" ");
17     }
18 }
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35

```

The right editor contains `App.java` with the following code:

```

src > App.java > App > main(String[])
You, hace 1 segundo | 1 author (You)
1 public class App {
2     Run | Debug
3     public static void main(String[] args) throws Exception {
4         //declarar
5         BucleFor bf;
6         BucleWhile bw;
7
8         //instanciar
9         bf = new BucleFor();
10        bw = new BucleWhile();
11
12         //Llamar el metodo
13         bf.signoAlterno();
14         bf.signoAlternoGenerativo();
15         bf.signoAlternoGenerativoSecond();
16         bf.escalera();
17
18         bw.signoAlterno();
19         bw.signoAlternoGenerativoSecond();
20     }
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35

```

Below the editors are tabs for PROBLEMAS, SALIDA, CONSOLA DE DEPURACIÓN, TERMINAL, PUERTOS, and GITLENS. The TERMINAL tab is selected, showing the output of the program:

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS GITLENS
While:
+ - + - + - + - +
While:
- + + - - + + + + -

```

The status bar at the bottom shows the path `C:\Users\Celeste\OneDrive\SAKI\programa\proyec2\Bucle2`, the current file `main`, the execution time `348ms`, and the time `1:30 PM`.

*Do While*

(xd)

## Apuntes semana 4

# Clase 7

## Paradigma

- Solventar problemas.

### 1. Conceptualización:

- Idea de lo que se quiere realizar, se puede dibujar, diseñar, etc. Tener un idea del resultado.
- Como realizarlo.
- ¿Que tengo? Ponerle nombre, concretar un significado.
- Solo los objetos se pueden hacer dos cosas: Características, acciones.
  - **Características:** propiedades. Almacenamiento de la información, cantidad, medir. (descripción clara del objeto). Recomendación: dibujarlo.
  - **Acciones:** métodos. Una acción es un verbo (jugar, correr, caminar...), que tenga sentido y vaya de acuerdo con las características mencionadas y con la conceptualización que se quiere, debe tener parámetros.
- No salir del tema, definir un límite, una parte de la información.
- No se invente.
- Solo lo necesario al momento de describir.
- Los parámetros podrían interpretarse como un límite.
  - **Parámetros:** dentro de las acciones Ej: Caminar(tiempo, lugar...) información extra que da las instrucciones, para que el programa haga lo que tu quieras. Se parece a las propiedades. Pero cambian.
- Definir si es público o privado/protector del objeto.
  - PUBLIC (+)
  - PRIVATE (-)
  - PROTEC ()
  - FRIENDLY (#)
- Definir si las características y los métodos son alguna de esas, eso depende del programador, tiene que tener una razón para saber cómo se va a comportar eso.

### 2. UML:

- **Significado:** Lenguaje de modelado unificado.
- Es el ¿Qué?
- Colocar en una clase. (Diagrama de clase)
- Ejemplo:

**+ CLASE**

---

Edad: float

---

---

tieneOjos: bool

---

---

tipoCabello: String

---

..

- Colocar las ideas, para que se identifiquen, para que estas empiezan a tener una forma mas solida, despues de colocar las caracteristicas se escriben los metodos.
- Ejemplo N.-2:

**+ CLASE**

---

Edad: float

---

---

tieneOjos: bool

---

---

tipoCabello: String

---

..

---

bailar(cancion: String; tiempoMin: int; ritmo: String): void/String/boolean

---

---

tocar(cosaobjeto: String; tiempoMin: int): boolena

---

---

saltar(altura: int; cantidad: int): void

---

..

- Void: no interesa
- String: que de un resultado
- Boolean: saber solo si cumplio o no cumplio.
- *Interaccion con otros objetos*: se necesita un evento (algo que pasa por algo externo, interaccion/comunicacion entre dos objetos) esto se puede añadir como un cuerpo u objeto mas, esos son los eventos.

**+ CLASE**

---

Edad: float

---

---

tieneOjos: bool

---

---

tipoCabello: String

---

..

---

bailar(cancion: String; tiempoMin: int; ritmo: String): void/String/boolean

---

---

tocar(cosaobjeto: String; tiempoMin: int): boolena

---

### + CLASE

---

salto(altura: int; cantidad: int): void

---

..

---

sentir (nombre: Hombre)

- Se pueden poner en lineas extra o allí mismo, depende del diseñador.
- se debe utilizar un herramienta para Diagrama de clases.

### 3. Código (Java)

- Diagrama y código deben estar iguales.
- *Ejemplo:* Propiedades.

```
public class Mujer {
    private float edad;
    public boolean tieneOjos;
    private String tipoCabello;
    ...
}
```

- *Ejemplo N.- 2:* métodos.

```
protected String bailar (String canción; int tiempoMin; String ritmo) {
    ...
    return "sjsjsmskdk..."
}
```

- *Deber*

- Conceptualizar un animal salvaje, a mano en una hoja.
- Debe tener 3 propiedades y 3 métodos.

## Clase 8

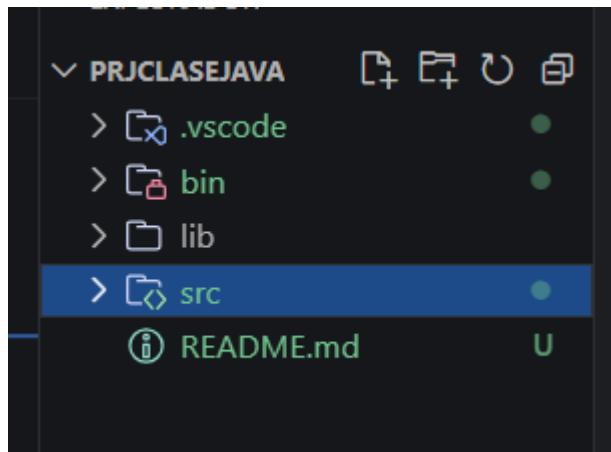
// después xd

## Clase 9

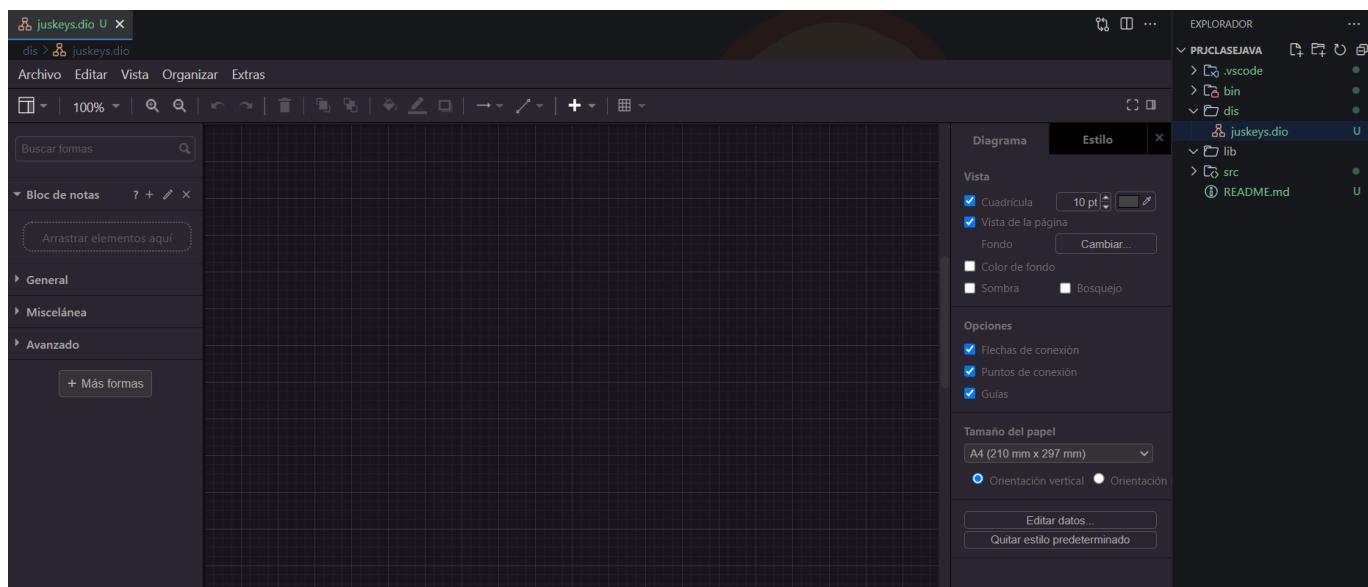
Comprobar que se tienen las siguientes extensiones:

- Herramientas:
  - drawn.io
  - excalidrawn
  - etc.

Abrir un nuevo proyecto, abrir solo la carpeta DEL PROYECTO, no más solo la carpeta, se tiene que ver así:



abri una pestaña de drawn.io

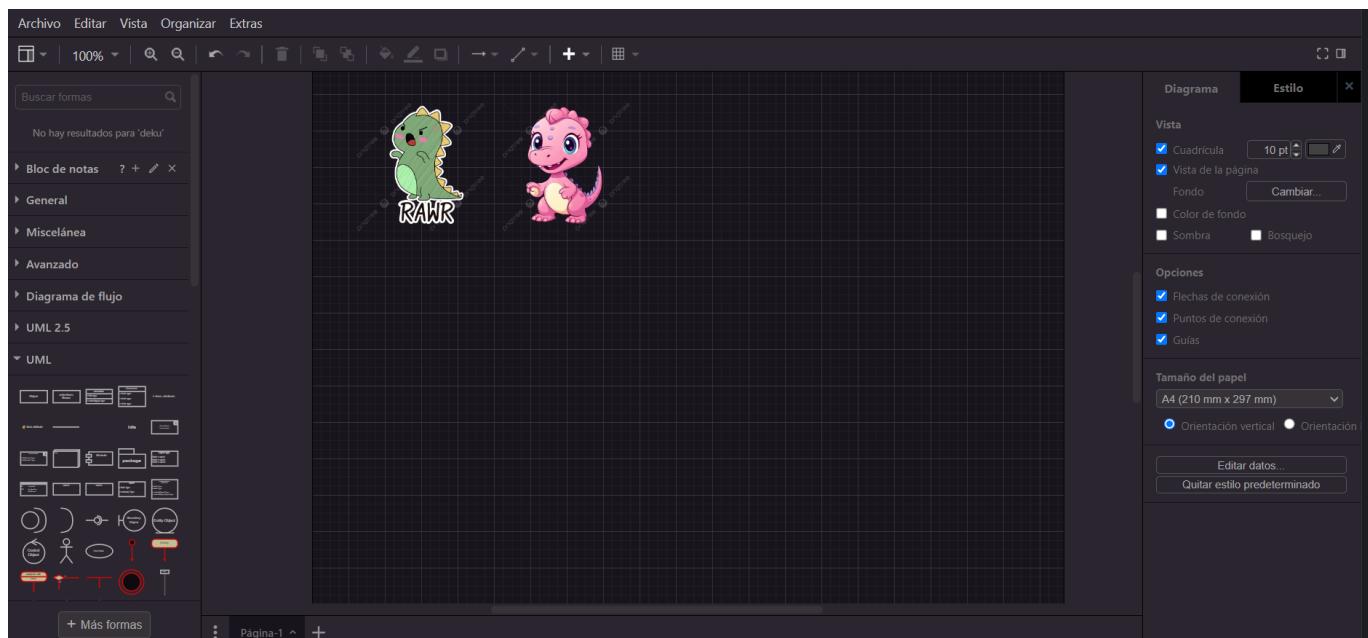


para los diseños abrir, en una nueva carpeta, nombre.dio (el "dio" es lo que hace que se transforme en ese formato)

### diagramas **Caso de Uso**

Es para definir lo que se quiere realizar, tener una imagen concreta

En este ejemplo se usaron dos imágenes:



Se tiene la imagen, (use) Donde se tienen varias ramas.

- relacion
- include
- externo

Pasa del diagrama de uso, interpretandolo, y pasandole a:

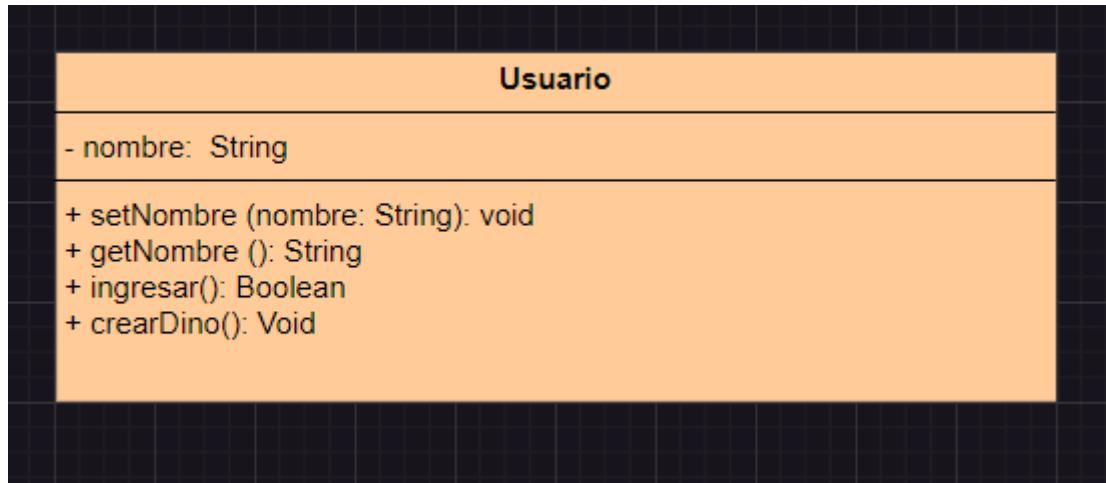
### Diagramas de Clase

- relacion
- asociacion
- herencia

(PARA EL PROYECTO SE DEBE TENER DIAGRAMA DE USO Y DGRAMA DE CLASE)

Comprobar que Java compile que la todo funcione

crear la imagen y pasar al UML



Tras eso pasarlo al codigo

```
▼ Click here to ask Blackbox to help you code faster
public class Usuario {
    private String nombre;

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

    public String getNombre() {
        return nombre;
    }

    public Boolean ingresar() {
        System.out.println("listo para el ingreso");
        return true;
    }

    public void crearDino() {
        System.out.println("crear un dino");
    }
}
```

y ver que eso compile

The screenshot shows an IDE interface with the following details:

- File:** Usuario.java
- Project:** Usuario
- Method:** setNombre(String)
- Code:**

```
1 public class Usuario {  
2     private String nombre;  
3  
4     public void setNombre(String nombre){  
5         this.nombre = nombre.toUpperCase();  
6     }  
7  
8     public String getNombre() {  
9         return nombre;  
10    }  
11  
12    public Boolean ingresar() {  
13        System.out.println("listo para el ingreso");  
14        return true;  
15    }  
16  
17    public void crearDino() {  
18        System.out.println("crear un dino");  
19    }  
20}  
21}
```

- Terminal Output:**

```
Celeste ➤ ② ③ ④ \OneDrive\SAKI\progra\prjClaseJava ➤ → (② ③ mai  
⚡Home ➤ /usr/bin/env C:\\Program\\ Files\\Java\\jre-1.8\\bin\\j  
e\\OneDrive\\SAKI\\progra\\prjClaseJava\\bin App  
●  
listo para el ingreso  
crear un dino
```

## Apuntes semana 5

### Clase 10

Ejemplo de como trabaja TUTI

Actor: cualquier cosa que lanza algo en el sistema, agente externo.

El actor o la persona dentro de este caso puede seleccionar/usuarios productos.

**Ejemplo:** En TUTI un cliente puede seleccionar un producto de los estantes. (siempre tiene que ir una frase completa).

**Ejemplo2:** pagar los productos seleccionados (ver quienes participan) quien recibe es el cajero (hay un sistema que ayuda a este trabajo, pero si no se le emplea bien igual tendra dificultades)

App = sistema.

Identificar que se usa en el proceso y las situaciones se pueden presentar.

- procesos de devoluciones.
- procesos de cancelación **Autorización de un supervisor**
- proceso enviar comprobante **electrónico**

Algo nuevo:

- productos en linea para envio a casa.
  - pago de productos seleccionado: obligatorio.
  - pagar con tarjeta de credito/debito: opcional.

### *Herencia*

aspectos comunes optimizar la gestion proteccion de informacion nombre; etc.

1.- conceptualizar. 2.- UML.

todo privado.

dinero en positivo.

el String es para las conversacion.

que poner cuando se tiene un valor numerico?

cuando se va a usar valor numerico? si se va a operar.

crear diseño = codigo respectivo al tema.

### *Generalizacion*

Palabras clave: *ES UN* (sube)

Plabra clave: *PUEDE SER* (descendente)

## **Clase 11**

- **Construcutor:** lleva el mismo nombre de la clase, siempre es publico o protec, esta por defecto(HASTA QUE SE LO DEFINE), se llama una sola vez. Inicializa las variables, se ejecuta automaticamente.
- **Sobrecarga:** recibe varios parametros.

## **Clase 12**

- **Las variables:** se pueden escribir con letras, iniciar en minusculas, no pueden contener espacios en blanco, puede tener un \_ (guion bajo), puede tener numeros incluso singo de dolar, pueden ser datos primitivos o no primitivos.
- **String** obtener un caracter, longitud, comparacion.
- **Scanner** leer un valor, entradas secuenciales, libreria, ayuda a escanear los datos que se ingresan en un flujo de informacion como con el teclado, los archivos, etc.
- **Arrays** se tienen varios modos, uno *EOF* se puede trabajar e varias formas.

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
=	x  = 3	x = x   3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

- **operadores**

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

```
main(String[] args)
```

- **Ejercicio** este sirve para validar el programa, y que pueda correr.

## Apuntes semana 6

---

### Clase 13

- *Interface*

abstraccion de los elementos.

UML Bosquejar los datos, diagramas de clases \* Estaticos (de clases, objetos, etc) \* Dinamicas (metodos, etc.) cosas involucradas en el proceso. hacer las pruebas necesarias.

Ejemplo:

al actor se le pone nombre bibliotecario, y las funciones que hace hacia el banco. otro "el estudiante" junto con las funciones que cumple y que pueden tener variables.

### *Herencia*

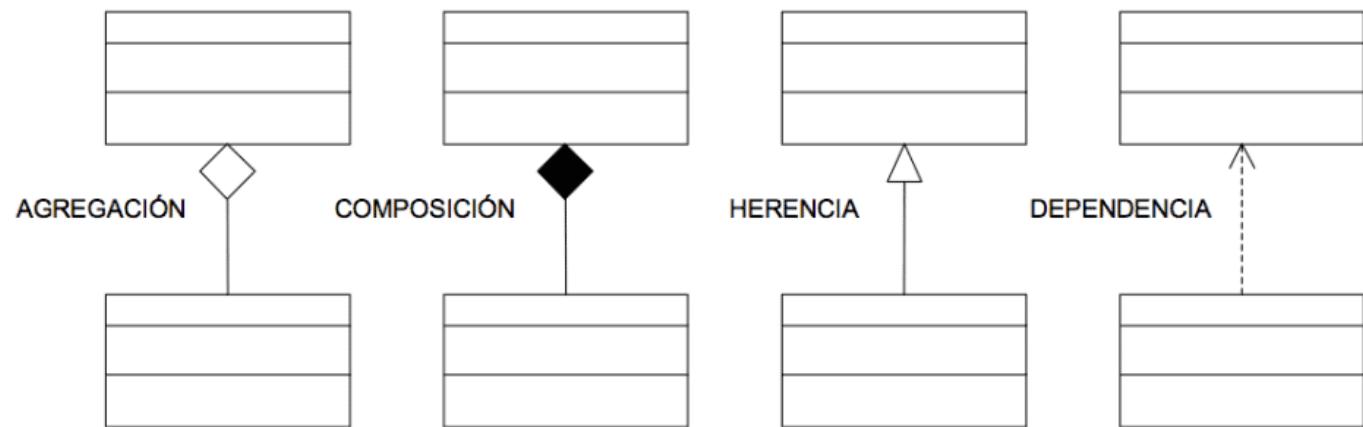
una clase puede heredar los atributos y metodos de otra clase.

especializacion entre actores y entre casos de uso.

se va especificando.

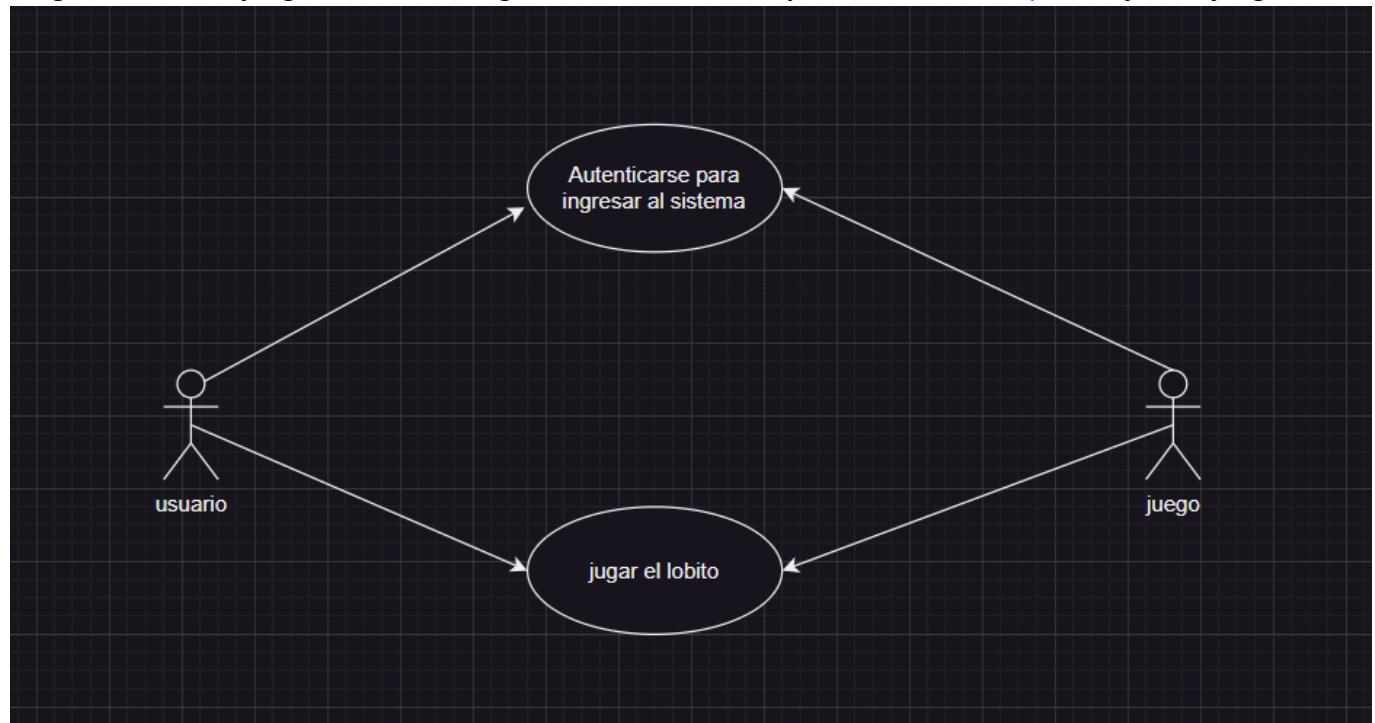
con el extend si se pueden poner mas de uno (es lo normal) con el include solo haya uno, puede tener otro, pero es complicado.

para busqueda.



## Clase 14

Diagrama de clase juego del lobo En el grafico se ve al usuario y al sistema (no los personajes del juego)



## Apuntes semana 7

---

### Clase 15

Scanner estatico: no puede clonarse, no se repite. sirve para declarar que solo se tiene un recurso o aparato (Ej.- el teclado)

Scanner dinamico: se puede clonar, se puede repetir.

logeo para la aplicacion

```
public class Jugador {
    private String usuario;
    private String clave;
    private String nombre;

    public Jugador(){
        setClave(clave:"DekuTodo");
        setUsuario(usuario:"Celes");
        setNombre(nombre:"Celeste");
    }

    public boolean login () {
        String usuario = "";
        String clave = "";
        boolean sinLogeo = false;

        do {
            System.out.print("Ingresa el usuario: ");
            usuario = App.sc.nextLine();
            System.out.print("Ingresa la clave: ");
            clave = App.sc.nextLine();

            if (this.usuario.equalsIgnoreCase(usuario)
                && this.clave.equalsIgnoreCase(clave))
                return true;

            System.out.print("para salir (s): ");
            if (App.sc.nextLine().toUpperCase().equals("S"))
                sinLogeo = true;
        } while (sinLogeo);
        return false;
    }
}
```

comandos de escape: \r: borra y vuelve a imprimir, por ello aparece en la misma posicion, \n: salto de linea

refactorizar: reorganizar el codigo para que sea mas legible y eficiente. no se cambia la funcionalidad, solo se reorganiza.

```
private Short mostrarMenu(){ ...  
  
public void moverBarca(String individuo) {  
    if (vikingoEstaIzq)  
        for (int i = 0; i < rio.length(); i++) {  
            setBarcaRio(i, individuo);  
        }  
    else  
        for (int i = rio.length() ; i >= 0; i--) {  
            setBarcaRio(i, individuo);  
        }  
  
}  
  
private void setBarcaRio(int posicionBarca, String individuo) {  
    String personajeIzq = Arrays.toString(ladoIzq);  
    String personajeDer = Arrays.toString(ladoDer);  
    String rioBarca = "\r"  
        + personajeIzq  
        + ".".repeat(posicionBarca)  
        + barca.replace(target:"?", individuo)  
        + ".".repeat(rio.length() - posicionBarca)  
        + personajeDer;  
    System.out.print( rioBarca);  
    try {  
        Thread.sleep(millis:50);  
    } catch (InterruptedException e) {}  
}
```

## Clase 17

Revision del proyecto.

- Aspectos del proyecto
  - PRESENTACION
    - caratura debe contener nombres, tema: de que va a tratar, al idea y como funciona
    - no mas de 10 lineas que explique el objetivo, alcance del proyecto
    - prototipo del proyecto.
    - debe utilizar el dispositivo externo, ¿cuál es?
    - 5 páginas de presentación.
  - DIAGRAMA DE CASO
    - no tan grande.
  - DIAGRAMA DE CLASES

- crear las clases necesarias, diseño de software.
- CODIGO
  - Bosquejo de los diagramas de clase, no funcional, navegable.
- Para el dia miercoles 12/06/2024

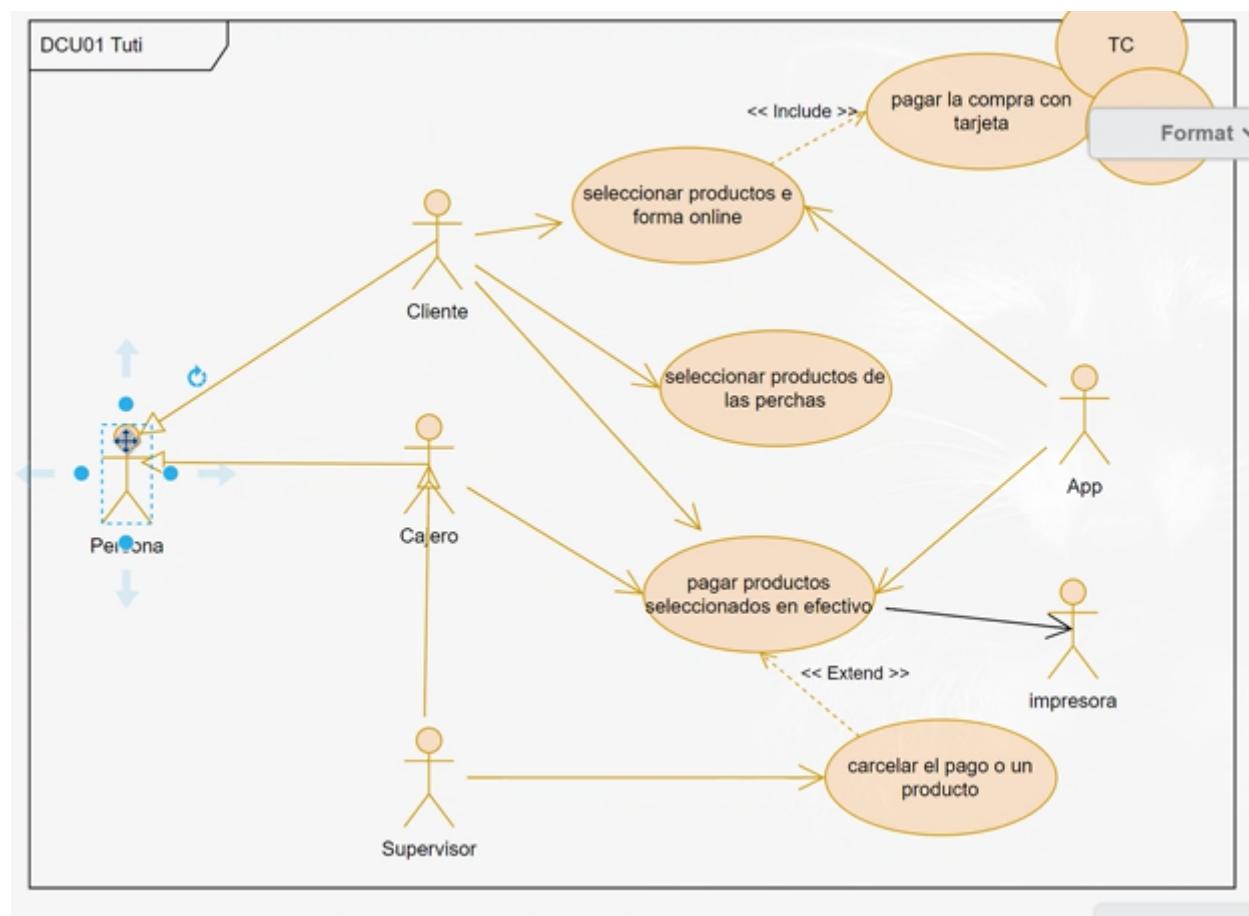
# Apuntes semana 8

## Clase 18

- Paquetes y abstractas

paquete para identificar temas de relacion.

- Clases abstractas

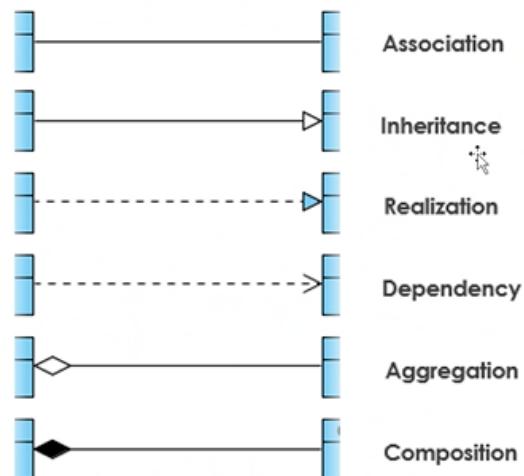


Examen

- PAQUETES
- ABSTRACTS
- INTERFACES
- HERENCIA

## Relaciones entre clases

Class Diagram Relationship Type	Notation
Association	
Inheritance	
Realization/ Implementation	
Dependency	
Aggregation	
Composition	



*asociacion*: relacion entre dos clases, donde dice que jamas va a existir uno sin el otro, ej: un propietario puede tener una mascota, pero una mascota no puede estar sin propietario.

*Interface/Realizacion/Implementacion*: son lo mismo los tres, no se instancian, no es su funcion heredar.

*Herencia*: una clase hija hereda las caracteristicas de una clase padre.

*Composicion*: una clase puede tener varias instancias de otra clase, ej: un coche tiene varias ruedas.

*Agregacion*: una clase puede tener varias instancias de otra clase, pero no necesariamente, ej: un coche puede tener varias ruedas, pero no necesariamente.

*Dependencia*: una clase puede utilizar otra clase, pero no es necesario, ej: un coche puede utilizar un GPS, pero no es necesario.

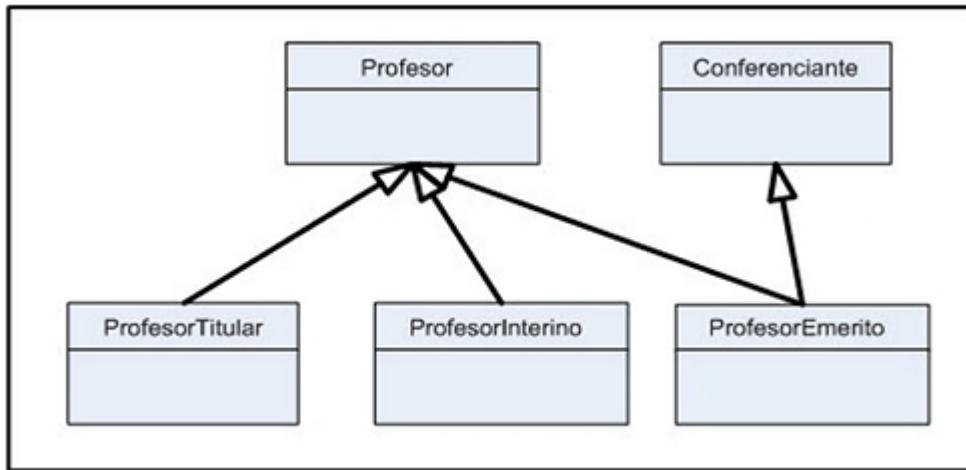
*interfaz(inombre)*: acciones comunes entre los entes, Ej: Icomportamientonatural, todos los metodos en la interfaz son publicos, por defecto.

en lugar de escribir dos veces el mismo procedimiento dos veces se prefiere aplicar una interfaz.

## Clase 19

- Realizacion
- Implementacion

son los mismo que una interfaz.



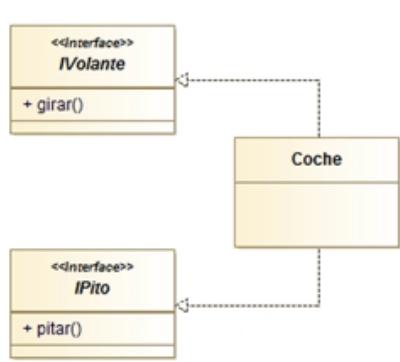
no se puede tener para uno mismo dos herencias, (ejemplo profesorEmerito) no es posible, cada hijo solo tiene un parente. pero es posible con una interfaz, patrones de diseño.

En una interface todo es abstracto y siempre es publico, no se puede hacer un new en el app.

leer el material\*

Ej:

↳ Implementación de Interfaces:



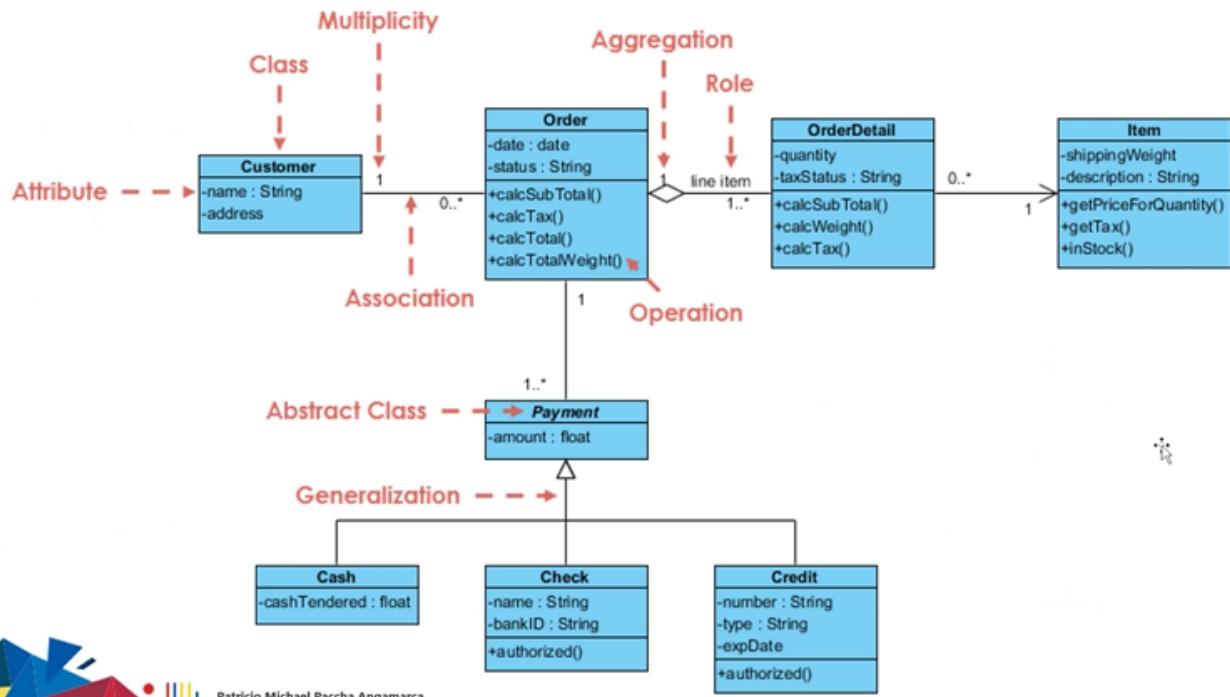
```

1  public interface IVolante {
2      public void girar();
3  }
4  public interface IPito {
5      public void pitard();
6  }
7  class Coche implements IVolante, IPito {
8      public void girar() {
9          System.out.println("¡Girando, girando!");
10     }
11     public void pitard() {
12         System.out.println("¡Pitando, pitando!");
13     }
14 }
  
```

Siempre una I al inicio para entender como una interface.

## Realización

Class Diagram Example: Order System



los numeros o asteriscos en las lineas significan cuantas variables deberia de declarar.