

TÉCNICAS DE PROGRAMACIÓN ORIENTADA A OBJETOS

16/04/2024 y 18/04/2024

UPN.EDU.PE

UNIDAD 1: FUNDAMENTOS DE PROGRAMACIÓN ORIENTADA A OBJETOS Y RELACIONES BÁSICAS ENTRE CLASES

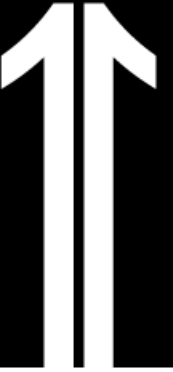


Sesión 4

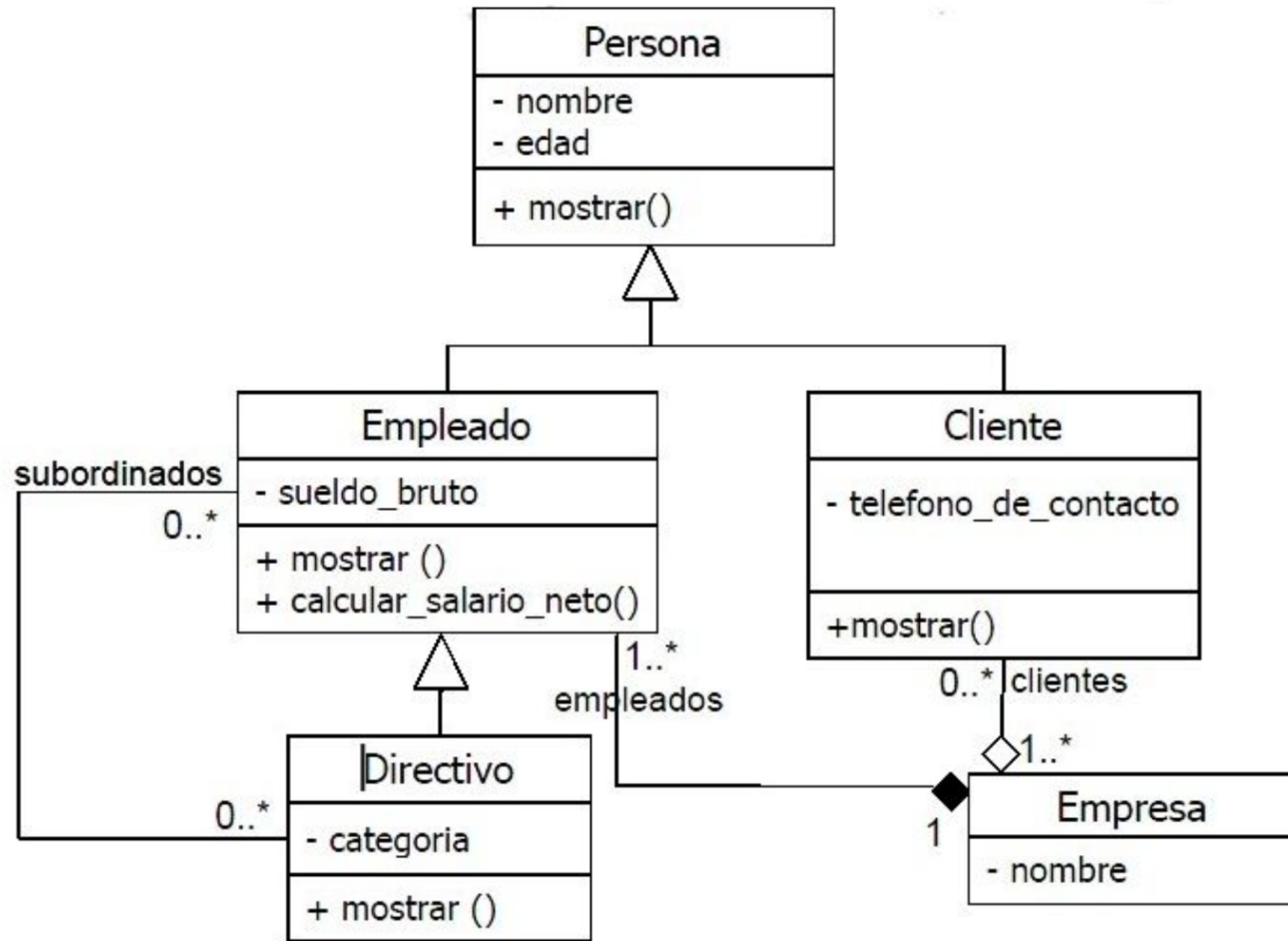
- Diagramas de clases y sus relaciones
- Relación Agregación y Composición

PRESENTACIÓN DE LA SESIÓN

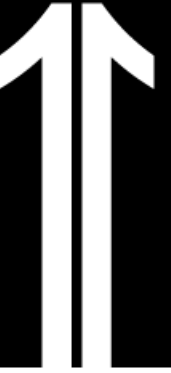
Logro de la Sesión y Temario



Al término de la sesión los estudiantes elabora un resumen de la Herencia con un ejercicio práctico.



REFLEXIONA



- ¿Qué son las relaciones entre clases ?
- ¿En qué se diferencia la asociación y composición?

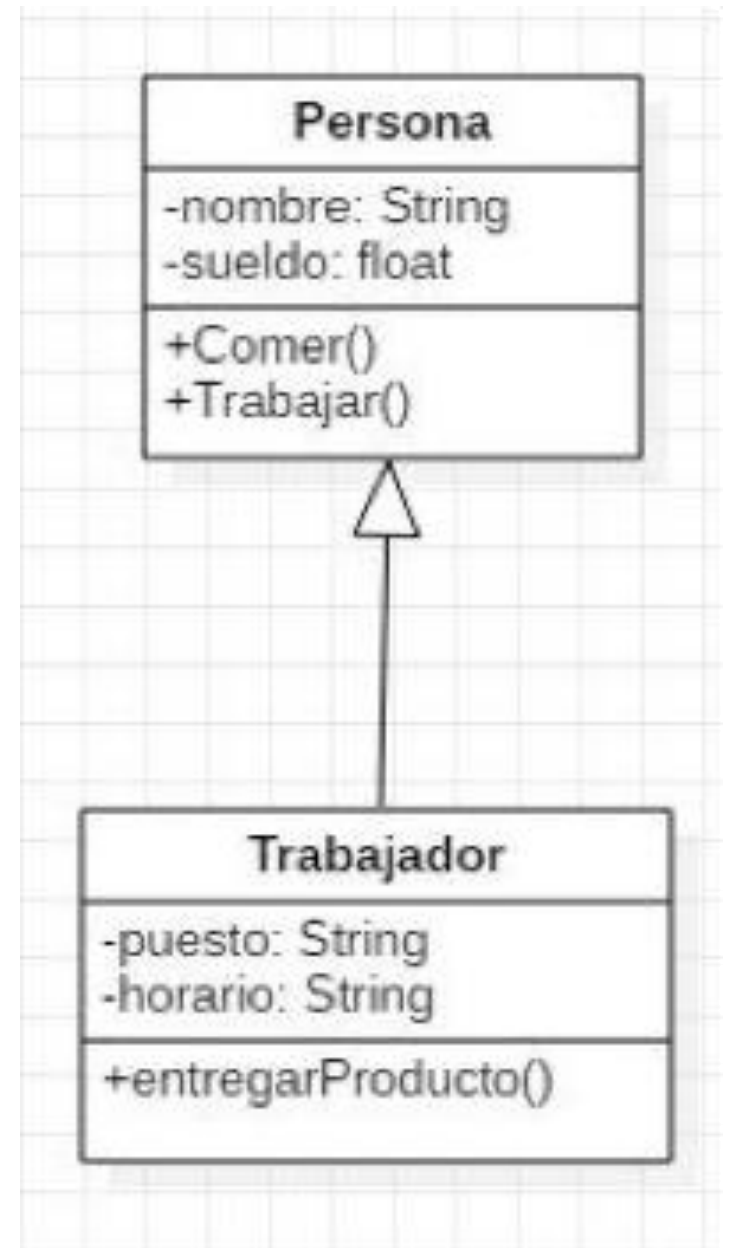
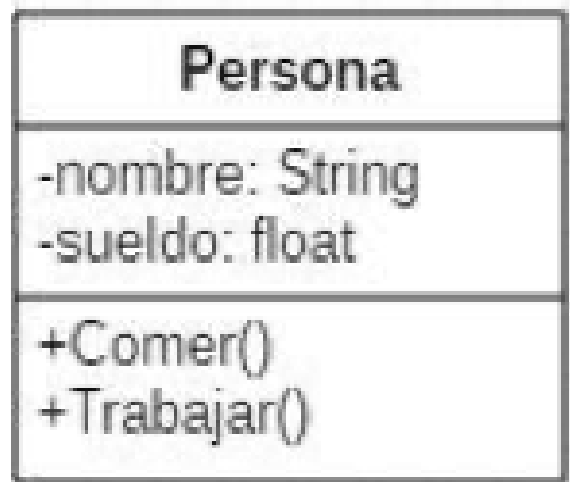


HERENCIA

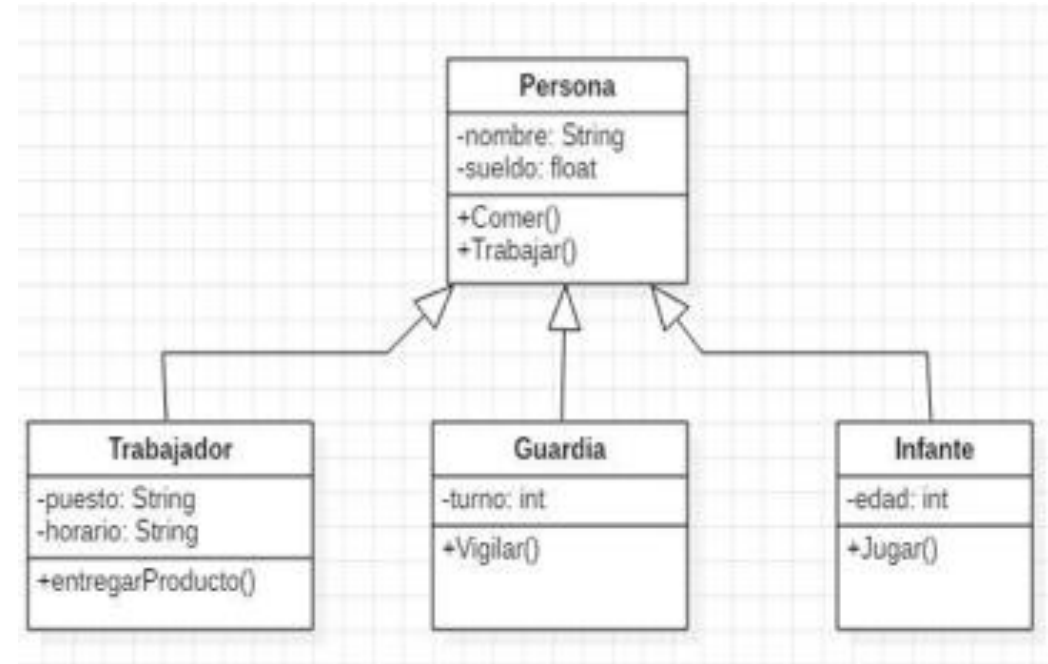
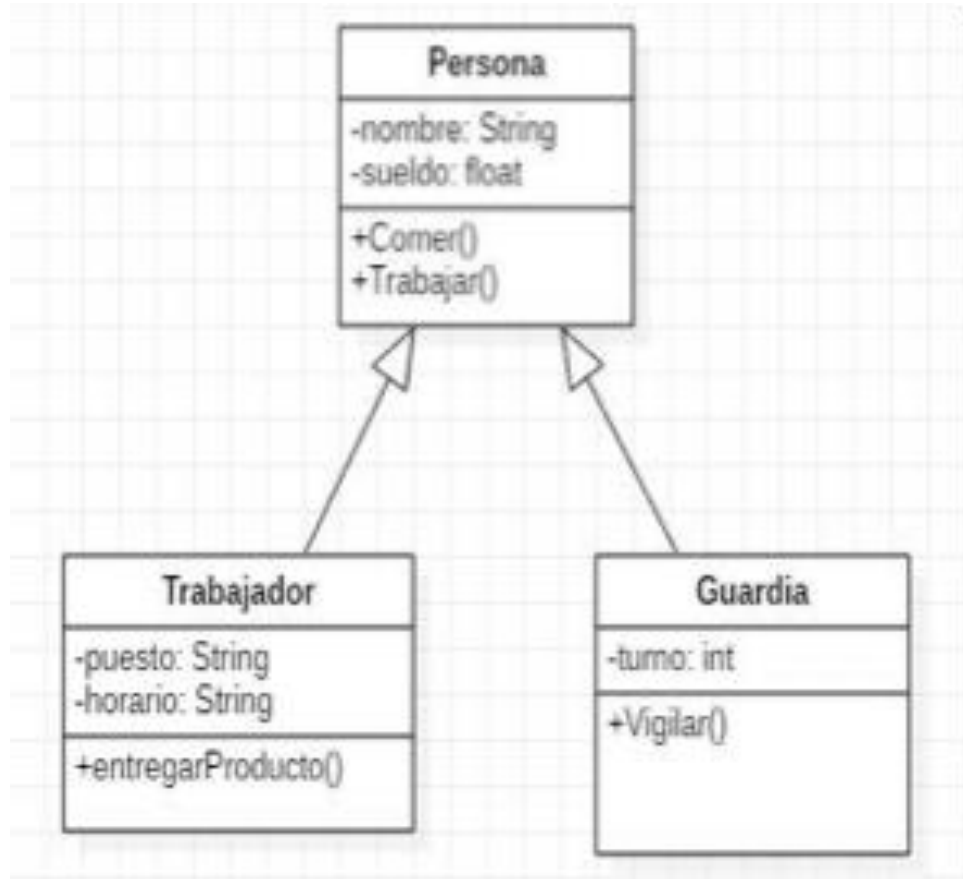
- Es uno de los pilares de la Programación Orientada a Objetos
- RELACIÓN “ ES UN ”
- PERMITE UNA CLASE QUE ADQUIERA LOS COMPORTAMIENTOS Y LOS ATRIBUTOS DE OTRA CLASE.
- SE DEBE ANALIZAR EN TÉRMINO DE ABSTRACCIÓN.



HERENCIA – UML:

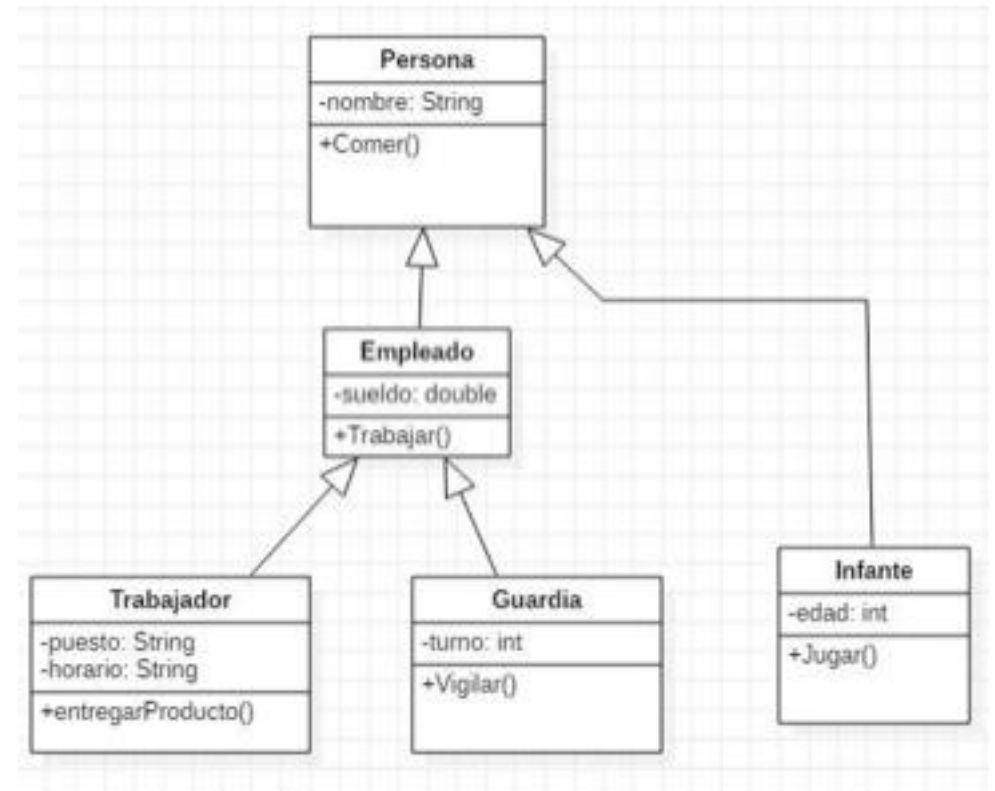
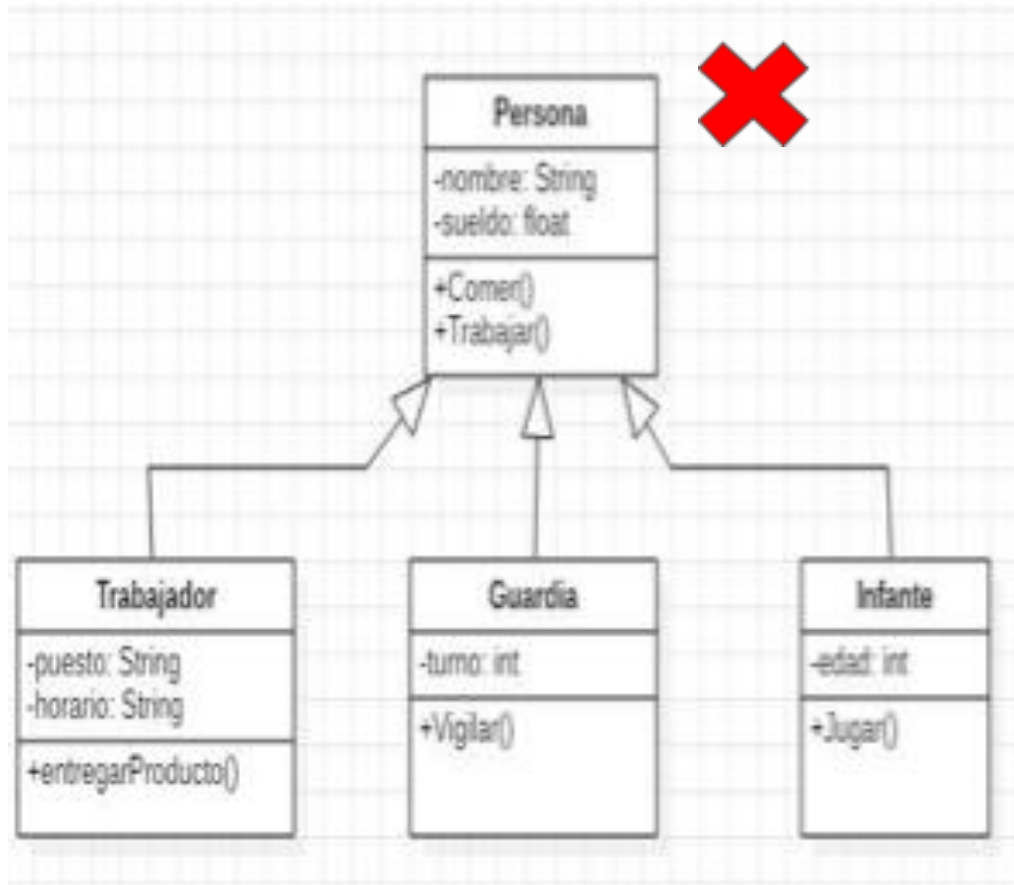


HERENCIA – UML:



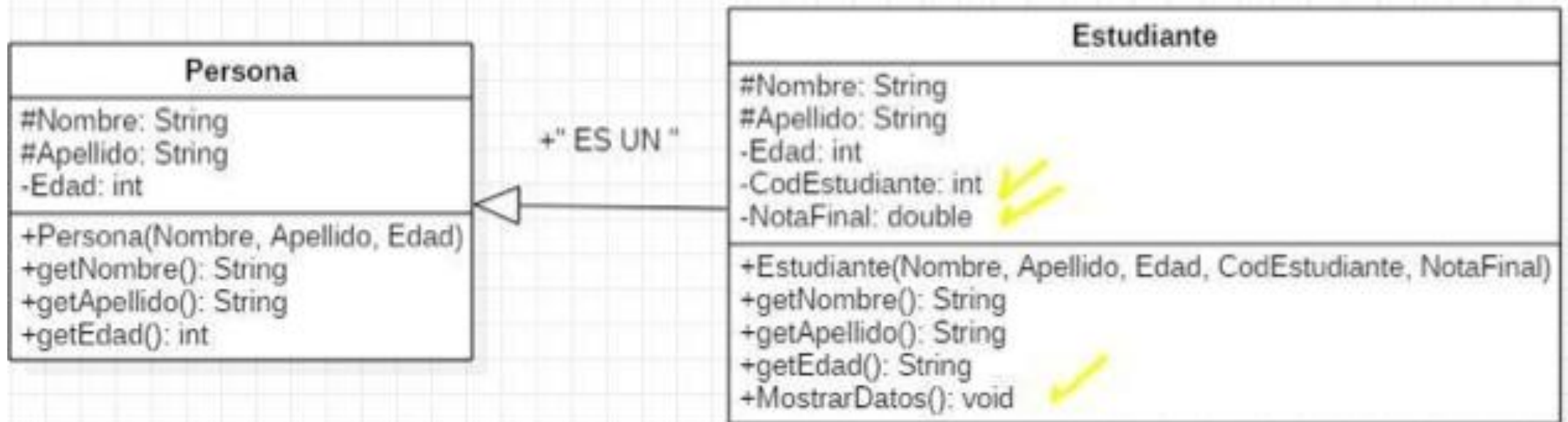



HERENCIA – UML:



EJEMPLO DE HERENCIA:

extends
super ()





```
package Herencia;
public class CPersona
{
    protected String nombre;
    protected String apellido;
    private int edad;
    public CPersona(String nombre, String apellido, int edad) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.edad = edad;
    }
    public String getApellido() {
        return apellido;
    }
    public int getEdad() {
        return edad;
    }
    public String getNombre() {
        return nombre;
    }
}
```



```
package Herencia;
```

```
public class CEstudiante extends CPersona{
```

```
    private int CodEstudiante;
```

```
    private double NotaFinal;
```

```
    public CEstudiante(int CodEstudiante, double NotaFinal, String nombre, String apellido, int edad) {
```

```
        super(nombre, apellido, edad); //llama al constructor de la clase padre
```

```
        this.CodEstudiante = CodEstudiante;
```

```
        this.NotaFinal = NotaFinal;
```

```
    }
```

```
    public void MostrarDatos()
```


```
    {
```

```
        System.out.println("Nombre: "+nombre+"\nApellido: "+apellido+"\nEdad: "+getEdad()+"
```

```
        + "\nCodigo: "+CodEstudiante + "\nNota Final: "+NotaFinal);
```

```
    }
```

```
}
```



```
package Herencia;
```

```
public class Principal {
```

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

```
        CEstudiante estudiante1 = new CEstudiante(12345, 18.5, "Nicolas", "Cruz", 35);  
        estudiante1.MostrarDatos();
```

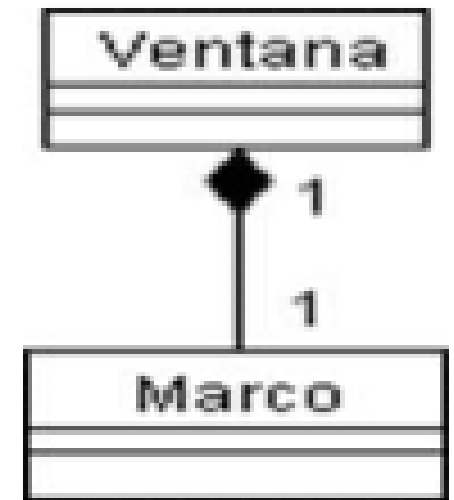
```
    }
```

```
}
```

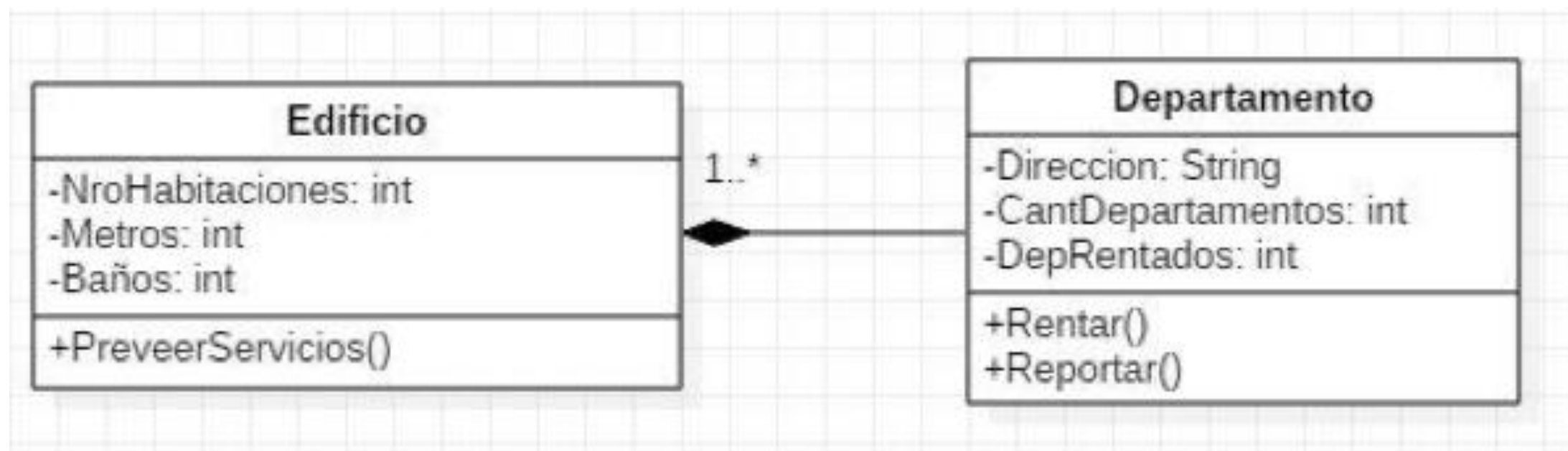


COMPOSICIÓN

- LA RELACIÓN ES DE “TIENE UN”
- ES CUANDO UNA CLASE TIENE OBJETOS DE OTRA CLASE.
- EL TIEMPO DE VIDA DEL OBJETO INCLUIDO ESTÁ CONDICIONADO POR EL TIEMPO DE VIDA DEL QUE LO INCLUYE.



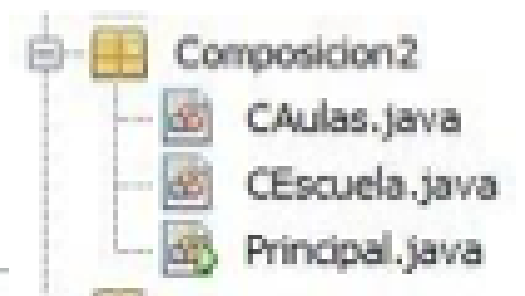
COMPOSICIÓN EN UML:



EJEMPLO DE COMPOSICIÓN:



EJEMPLO DE COMPOSICIÓN:



```
package Composicion2;
```

```
public class CAulas
```

```
{
```

```
    private String nombre;
```

```
    private int CantAlumnos;
```

```
    public CAulas(String nombre, int CantAlumnos) {
```

```
        this.nombre = nombre;
```

```
        this.CantAlumnos = CantAlumnos;
```

```
    }
```

```
    @Override
```

```
    public String toString()
```

```
    {
```

```
        String mensaje=" ";
```

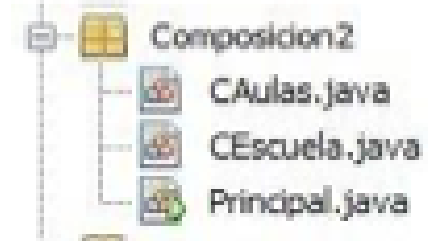
```
        mensaje= "El aula de" + nombre + " tiene "+CantAlumnos + " alumnos";
```

```
        return mensaje;
```

```
    }
```

```
}
```

EJEMPLO DE COMPOSICIÓN:

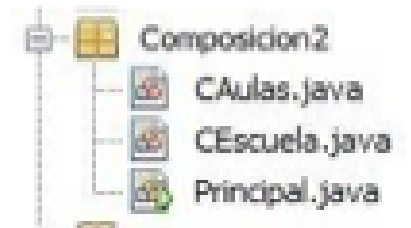


```
package Composicion2;

import java.util.Scanner;
public class CEscuela
{
    private String nombre;
    private CAulas aulas[]=new CAulas[3]; //Composicion

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

EJEMPLO DE COMPOSICIÓN:



```
public void ColocaAulas()
{
    Scanner obj=new Scanner(System.in);
    String nombre= " ";
    String dato=" ";
    int cantidad=0;

    for( int i=0;i<aulas.length;i++)
    {
        System.out.println("Ingrese el nombre del aula: ");
        nombre=obj.nextLine();

        System.out.println("Ingrese la Cantidad de alumno: ");
        dato=obj.nextLine();
        cantidad=Integer.parseInt(dato);
        aulas[i]=new CAulas(nombre,cantidad);
    }
}
```

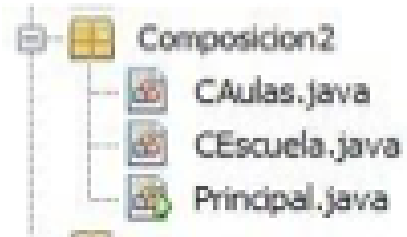
EJEMPLO DE COMPOSICIÓN:



```
@Override
public String toString()
{
    String mensaje= " ";
    mensaje="Bienvenidos a la Escuela "+ nombre+"que tiene: \n";
    for(int i=0;i<aulas.length;i++)
    {
        mensaje+=aulas[i].toString()+ "\n";
    }
    return mensaje;
}
```

```
}
```

EJEMPLO DE COMPOSICIÓN:



```
package Composicion2;

public class Principal {

    public static void main(String[] args)
    {
        //Creamos la escuela
        CEscuela escuela1=new CEscuela(" UCV ");
        //Creamos el objeto de aula que existe en Escuela
        escuela1.ColocaAulas();

        //Mostramos el objeto compuesto
        System.out.print(escuela1);
        //Destruimos escuela1
        escuela1=null;

        System.out.println("\n"+escuela1);

    }
}
```

Output - Principal (run)

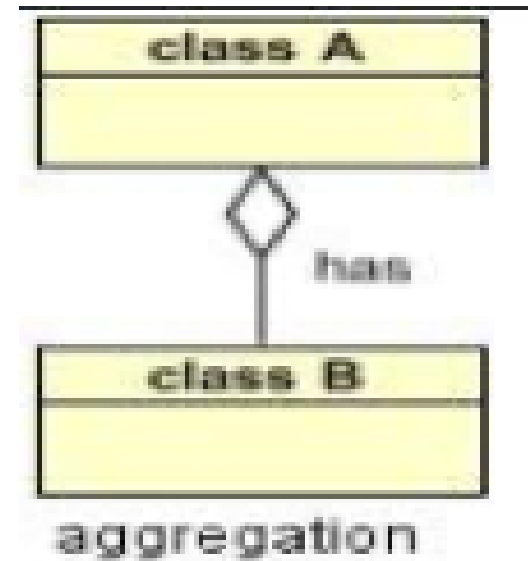
```
run:
Ingrese el nombre del aula:
B201
Ingrese la Cantidad de alumno:
34
Ingrese el nombre del aula:
B109
Ingrese la Cantidad de alumno:
40
Ingrese el nombre del aula:
B304
Ingrese la Cantidad de alumno:
39
Bienvenidos a la Escuela UCV que tiene:
El aula deB201 tiene 34 alumnos
El aula deB109 tiene 40 alumnos
El aula deB304 tiene 39 alumnos

null
BUILD SUCCESSFUL (total time: 21 seconds)
```

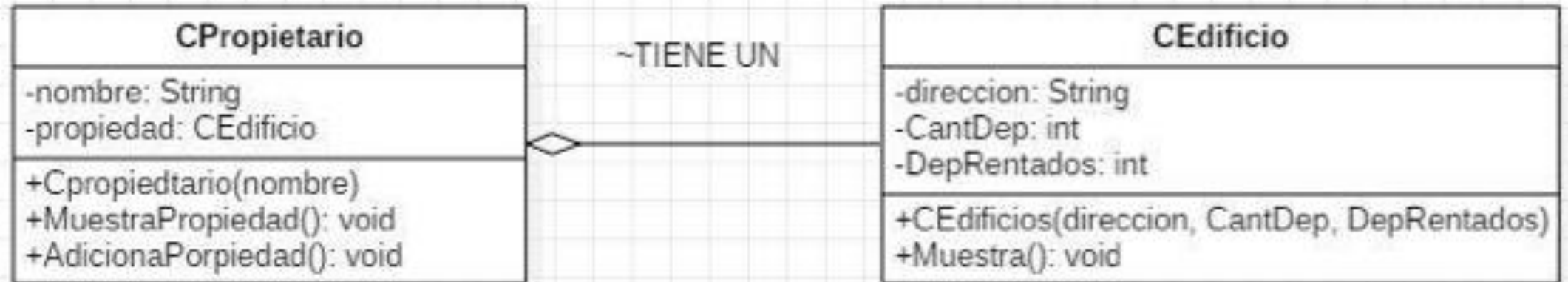


AGREGACIÓN

- LA RELACIÓN ES DE “TIENE UN”.
- A PESAR DE TENER LA MISMA RELACIÓN DE UNA COMPOSICIÓN, LA DIFERENCIA RADICA EN EL TIEMPO DE VIDA DEL OBJETO QUE ES INCLUIDO, ES INDEPENDIENTE DEL QUE LO INCLUYE.



EJEMPLO DE AGREGACIÓN:



EJEMPLO DE AGREGACIÓN:



```
package Agregacion;

public class CEedificio {
    private String direccion;
    private int CantDepa;
    private int DepRentado;

    public CEedificio(String direccion, int CantDepa, int DepRentado) {
        this.direccion = direccion;
        this.CantDepa = CantDepa;
        this.DepRentado = DepRentado;
    }

    public void Muestra()
    {
        System.out.println("Tiene un edificio ubicado en "+direccion+
            " y posee "+CantDepa + " departamentos"+ " de los cuales "
            +DepRentado+" estan rentados" );
    }
}
```



```
public class CPropietario
{
    private String nombre;
    private CEdificio propiedad=null; // No es una instancia de clase

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

    public void MuestraPropiedad()
    {
        if(propiedad!=null)
        {
            System.out.println("El propietario "+ nombre);
            propiedad.Muestra();
        }
        else
        {
            System.out.println(nombre + " aun no tiene una propiedad");
        }
    }
}
```



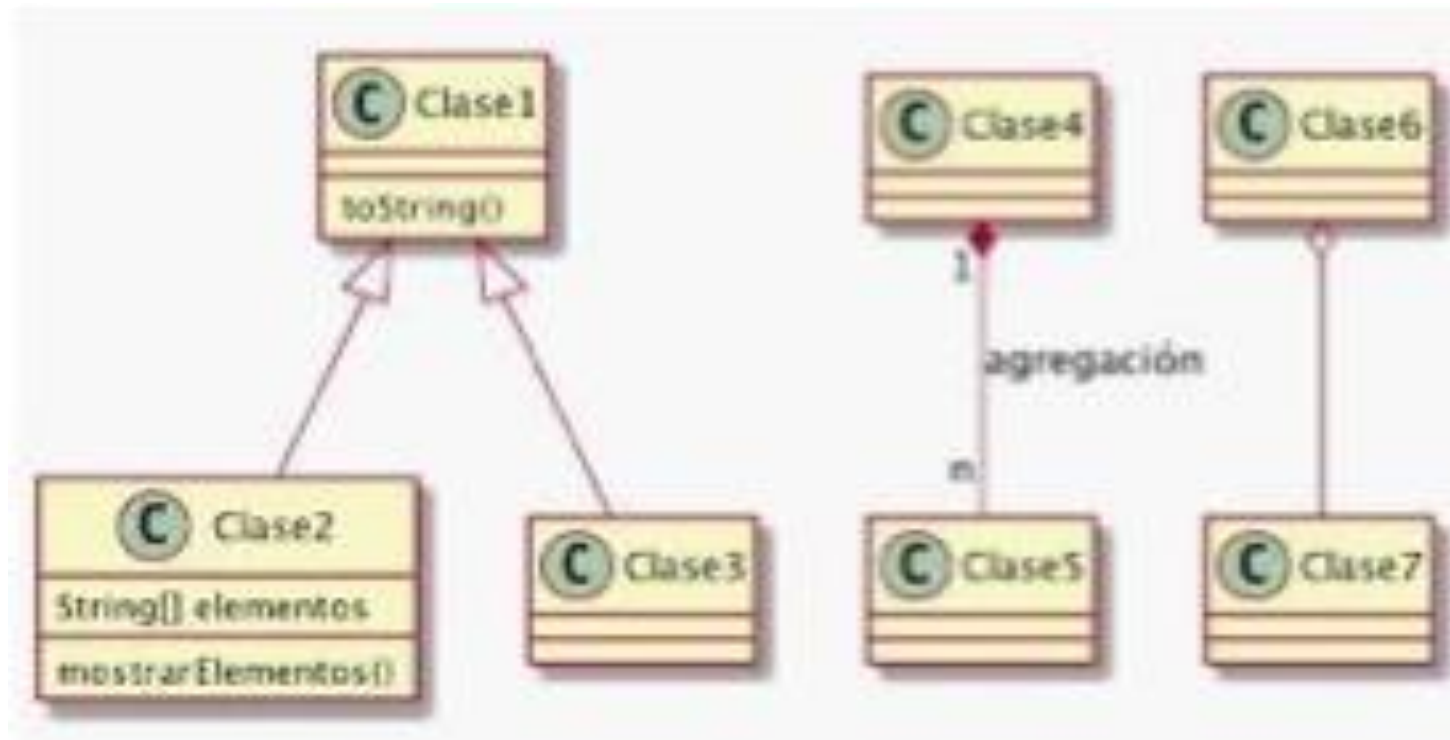
EJEMPLO DE AGREGACIÓN:



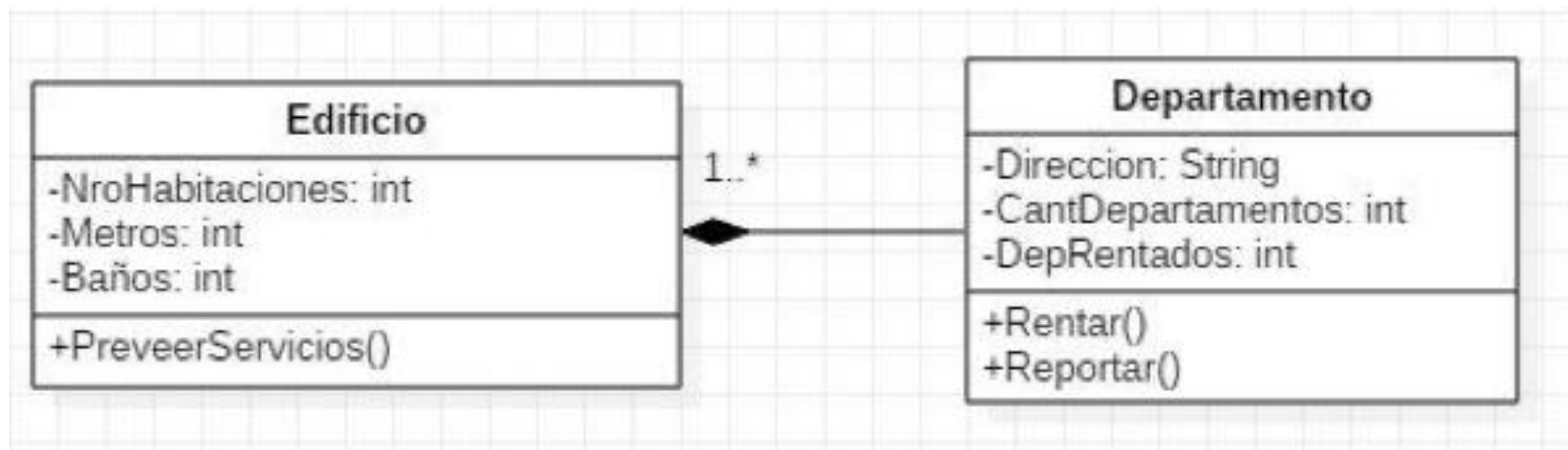
```
public void AdicionaPropiedad( CEdificio edificio1)
    //Aqui se verifica "LA AGREGACIÓN"
{
    if(edificio1 !=null)
    {
        propiedad=edificio1;
    }
}
}
```

CARDINALIDAD EN UML:

- Número de objetos que participan en una relación entre clases...



CARDINALIDAD EN UML:





NOTACIÓN DE CARDINALIDAD:

Multiplicidad	Significado
1	Uno y sólo uno
0..1	Cero o uno
N..M	Desde N hasta M
*	Cero o varios
0..*	Cero o varios
1..*	Uno o varios (al menos uno)

EJERCICIO TI

Caso: EMPRESA LOGÍSTICA “FIFO”

La empresa logística “FIFO” cuenta con una flota de camiones que realizan recorridos diarios a puntos de distribución para dejar y recoger mercadería. Sin embargo, debido a malas programaciones en el diseño de las rutas, la empresa ha generado pérdidas en combustible y tiempo. Por ello se busca que la empresa pueda generar ahorros programando de manera más eficiente sus rutas, considerando las distancias entre ellos.



Se cuenta con las coordenadas (longitud y latitud) del punto de partida y de cada punto de distribución, así como los nombres de estos últimos.

En base al caso planteado, se pide lo siguiente:

1. Elaborar un diagrama de clases del caso.
2. Diseñar un sistema con una interface visual que permita determinar cuál será el punto a cubrir por cada camión, considerando que cada camión irá a un solo punto. El sistema deberá:
 - a. Permitir al usuario registrar los puntos de distribución (3) que se tomarán en cuenta para la asignación de las rutas (cada punto tiene: nombre, longitud y latitud)
 - b. Las rutas se asignarán considerando inicialmente la más alejada, luego la intermedia y finalmente la más cercana.
 - c. Mostrar el punto al que cada camión se dirigirá ordenándolo de mayor a menor distancia. Por ejemplo:
Camión 1 -> Punto 2 (más lejano)
Camión 2 -> Punto 1 (intermedio)
Camión 3 -> Punto 3 (más cercano)
 - d. Mostrar la distancia total a recorrer durante el día por los tres camiones (suma de las distancias recorridas por cada uno)
 - e. Consideraciones:
 - i. Cada unidad de distancia calculada en base a las coordenadas de cada punto de distribución equivale a 100 metros.
 - ii. El cálculo parte de un punto de salida, cuyas coordenadas son ingresadas al sistema.

Punto de salida

Long Lat

Puntos de distribución

Ruta sugerida

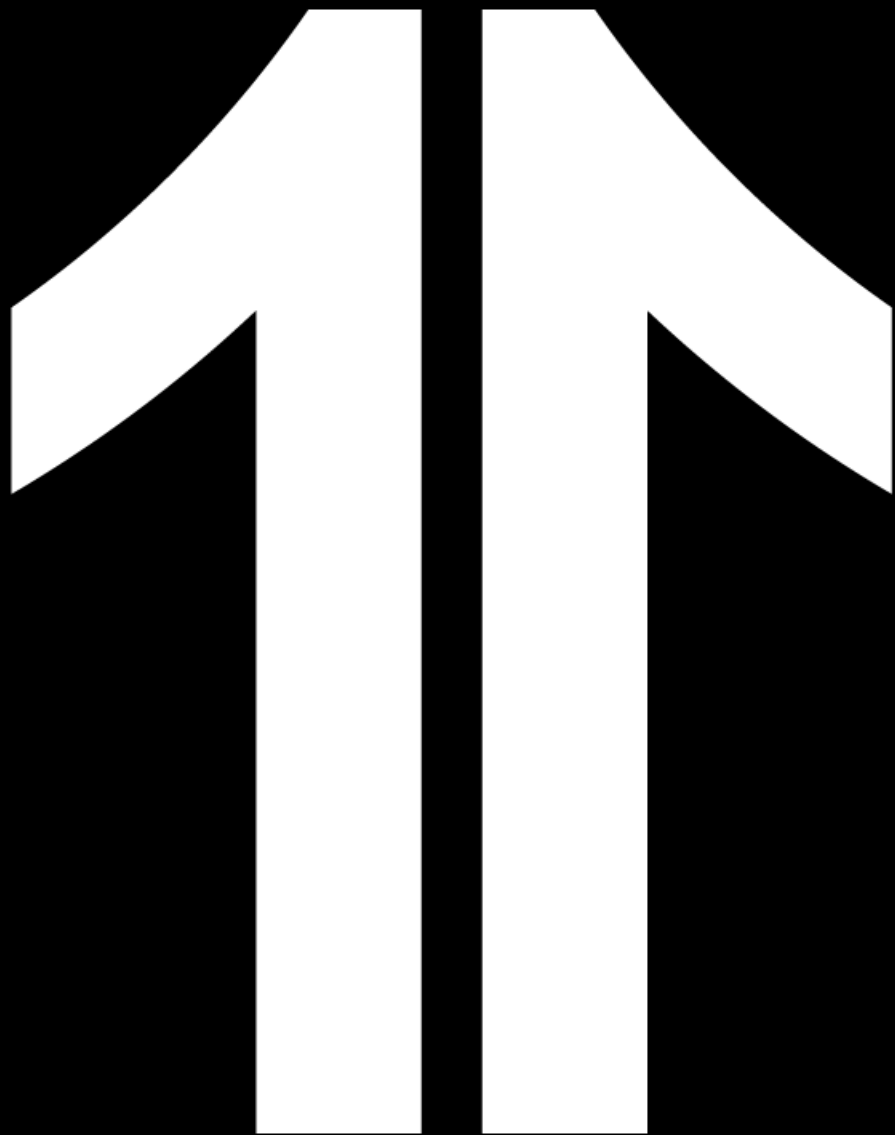
Agregar punto

Distancia total km

Long Lat

Nombre

Calcular ruta



GRACIAS

