Nombre Alumno/s: Reyes, Kevin Josue **DNI:** 45939556
Nombre Profesor: Aldo Mettini – Andrea Airaldi
Grupo Laboratorio: 3 **TP:** 3 **Fecha de entrega:** 07-09-2025

Informe de Laboratorio

Introducción

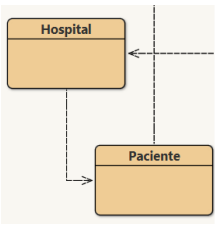
Este trabajo práctico tuvo como objetivo aplicar los conceptos centrales de la Programación Orientada a Objetos estudiados en la teoría, especialmente los vinculados con el conocimiento entre objetos y la interpretación e implementación de diagramas UML. También, se continuó reforzando los conceptos de encapsulamiento, abstracción, clase, objeto y mensajes.

Desarrollo

Lo novedoso de este trabajo práctico fue aprender y aplicar el concepto de **conocimiento entre objetos**. Comprendimos que los objetos no existen de manera aislada, sino que necesitan conocerse para poder colaborar. Este conocimiento puede darse de distintas formas: mediante **variables de instancia** (un producto que conoce al laboratorio que lo fabrica), **parámetros** (un método que recibe otro objeto como argumento), **variables temporales** (objetos usados solo dentro de un método) y la **seudo-variable** *this* (cuando el objeto se refiere a sí mismo). Estos casos nos ayudaron a entender que la interacción entre objetos es lo que hace funcionar al sistema.

Ej. 1: Conocimiento entre objetos mediante variables de instancia

```
public class Hospital {
    //Atributos
    private String nombreHospital;
    private String nombreDirector;
    private Paciente paciente;
```



Un objeto de tipo *Hospital* conoce a un objeto de tipo *Paciente*.

Ej. 2. Conocimiento entre objetos mediante variables temporales.

```
public boolean esCumpleaños() {
    Calendar hoy = new GregorianCalendar();

    return (hoy.get(Calendar.MONTH) == this.getFechaNacimiento().get(Calendar.MONTH) &&
            hoy.get(Calendar.DAY_OF_MONTH) == this.getFechaNacimiento().get(Calendar.DAY_OF_MONTH));
}
```

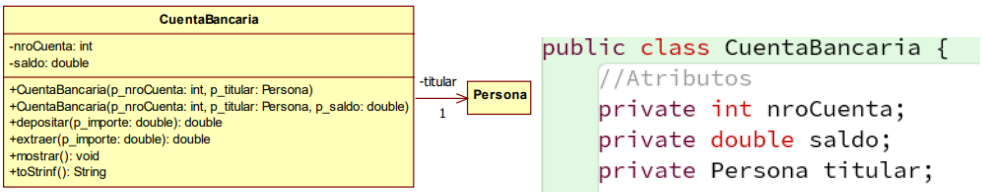
Ej. 3: Conocimiento entre objetos mediante parámetros

```
public void consultaDatosFiliatorios(Paciente p_paciente) {
    System.out.println("Hospital: " + this.getNombreHospital() + "\t" +
                       this.getNombreDirector());
    System.out.println("-----------------------------------------------------");
    p_paciente.mostrarDatosPantalla();
}
```

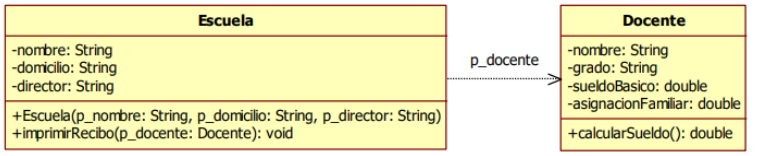
El método *consultaDatosFiliatorios* recibe como argumento a un objeto tipo *Persona*. Además, se observa la seudo-variable *this* haciendo autoreferencia a un método propio.

A partir de los **diagramas UML** también profundizamos en las **relaciones entre clases**. El uso de los gráficos nos permitió interpretar de manera visual cómo se conectan los objetos en un sistema:

- La **cardinalidad** (* o n) nos indicó cuántas instancias de una clase se vinculan con otra. Por ejemplo, la cardinalidad 1 en *CuentaBancaria* indica que tiene 1 titular de tipo *Persona*.
- Una **línea continua** representó una asociación fuerte entre clases, que luego se tradujo en atributos dentro del código.

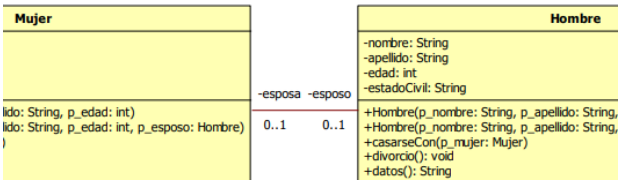


- Una **línea punteada** señaló una dependencia, una relación más débil que indica que una clase necesita a otra solo en determinado momento (como cuando un objeto aparece en un parámetro).

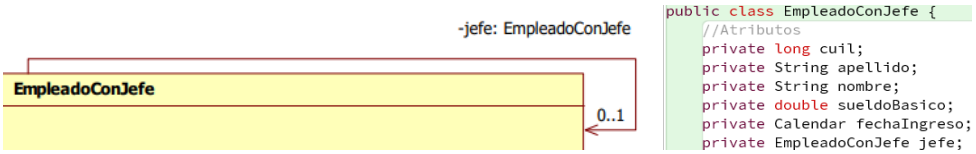



```
public void imprimirRecibo(Docente p_docente) {
    System.out.println ("\n***RECIBO DE SUELDO***");
    System.out.printf("Escuela: %s \t Domicilio: %s \t Director: %s \n",
                      this.getNombre(), this.getDomicilio(),
                      this.getDirector());
```

- Las relaciones **sin flecha al final** marcaron asociaciones bidireccionales, donde ambos objetos se conocen. En este caso la cardinalidad 0.1 indica que pueden o no conocerse.



- Y en los casos donde la flecha volvía a la misma clase, comprendimos que se trataba de una **asociación reflexiva**, es decir, una clase que se vincula consigo misma.



UML es una guía que después trasladamos a código, creando atributos, parámetros o métodos según la relación indicada.

Además de estos nuevos aprendizajes, continuamos aplicando y reforzando los conceptos ya conocidos: el **encapsulamiento**, al declarar atributos como privados y controlar su acceso con métodos; la **abstracción**, al modelar entidades del mundo real resaltando sus características esenciales; y los **mensajes**, que expresan la comunicación entre objetos y se implementan en Java como métodos. Si bien ya los conocíamos, al usarlos junto con el conocimiento entre objetos y las relaciones de UML pudimos apreciar más claramente cómo cada principio se conecta con el otro.

Conclusión

Este TP nos mostró que un sistema orientado a objetos se construye como una red de entidades que encapsulan su estado, ofrecen comportamientos y colaboran con otros a través de mensajes, gracias al conocimiento que tienen entre sí.