

GUÍA DE PRÁCTICA DE LABORATORIO Nro 01

EXPERIENCIA CURRICULAR: Metodología de Programación

DOCENTE: Mg. Ing. Ivan Petrlik Azabache / Ing. Eric Gustavo Coronel Castillo

SESIÓN: 1

DESARROLLO DE LA PRÁCTICA

CAPACIDAD A TRABAJAR:

Construye Programas básicos usando clases y objetos.

CONCEPTOS:

Programación Orientada a Objetos

Podríamos definir la Programación Orientada a Objetos (**POO** u **OOP** en inglés) como una forma de programar en la que se plantean las cosas intentando realizar una asociación con objetos de la vida real, y expresándolas mediante un conjunto determinado de técnicas de programación.

Por ejemplo, si pensamos en un coche nos daremos cuenta de que todos tienen en común determinadas características (marca, modelo, color, cilindrada, etc.) y realizan las mismas acciones (arrancar, acelerar, frenar, apagar, etc.).

El uso de una buena POO facilita enormemente la modularidad de un programa informático permitiendo dividirlo en partes más pequeñas e independientes, así como la detección y depuración de errores, su posterior mantenimiento, y la reutilización del código fuente en otros programas informáticos.

Dentro de este tipo de lenguajes de programación, los más conocidos son **C++** y **Java**, y cabe destacar que no todos ellos implementan las mismas características definidas en dicha metodología.



Si nunca has estudiado sobre **POO** y tras leer lo referente a dicha técnica de programación en el presente te curso te da la impresión de ser algo demasiado complicado no te preocupes... no serás el primero ni

el último a quien le sucede al principio (incluso teniendo experiencia en otros tipos de lenguajes de programación).

Para comprenderlo todo mejor, consulta los enlaces en los que explicamos cómo hacer las cosas en determinados lenguajes de programación.

La ventaja de utilizar Objetos en lugar de simples Funciones (usadas en la programación estructurada) es que si bien éstas también pueden ser reutilizadas, en caso de tener que realizar un cambio en su comportamiento no sería necesario volver a repetir el código fuente (bastaría con crear otra Clase que herede de la ya existente, y **redefinir los métodos** existentes o añadir otros nuevos).

Clases y Objetos

Podemos entender un Objeto como la representación de una **entidad** de la vida real con la cual podremos interactuar en el programa.

Antes de poder **crear un Objeto** es necesario crear una definición del mismo, por lo que primeramente deberemos **crear una Clase**, la cual contendrá como **Miembros**:

- **Propiedades** / **Atributos**: variables que describen características del Objeto o estados del mismo.
- **Métodos**: los **Métodos** se crean de forma parecida a las funciones, y son usados tanto para asignar o devolver el valor de las **Propiedades**, como para describir la forma en que se comporta el Objeto.

Según el caso, no todos los Miembros de una Clase deben poder ser accesibles desde fuera de ella: para ocultarlos usaremos lo que se conoce como encapsulamiento, pudiendo ser:

- **public**: se puede acceder a ellos desde cualquier lugar en el que sea posible acceder a la Clase, y también desde las que **heredende** ella.
- **private**: sólo es posible acceder a ellos usando los método proporcionados por la propia Clase (tampoco pueden acceder directamente las clases que hereden de ella).
- **protected**: accesibles desde las clases que hereden de ella, y desde otras que estén en el mismo **package** o paquete.

En este punto, un término que debes conocer es el de **polimorfismo**, que se refiere al hecho de que usando un mismo nombre podemos obtener comportamientos (formas) diferentes, lo que se consigue por medio de la **sobrescritura** y **sobrecarga** de métodos y el uso de **interfaces**. A continuación explicamos el primero de ellos y algo más adelante las otras dos formas de implementarlo.

Las clases deben tener un método denominado **constructor**, a partir del cuál se crearán **Instancias** / **Objetos**.

Es posible que una clase tenga más de un constructor (con el mismo nombre) siempre y cuando tengan parámetros de entrada diferentes. Ello se denomina **sobrecarga de métodos** (también es aplicable a sus otros **métodos**).

Una vez explicado todo esto veamos ahora un ejemplo de una clase creada en lenguaje Java.

[Expand Code](#)

Coche.java:

[Expand Code](#)

Al ejecutar el programa veríamos lo siguiente:



```
Output - PatronesJava (run)
run:
Puertas: [2]
Marchas: [3]
-----
Puertas: [4]
Marchas: [5]
Bastidor: [WAUZZZ8EZ1A000000]
Tracción: [DELANTERA]
=====
Puertas: [2]
Marchas: [6]
Bastidor: [VSSZZZ5PZ4R000000]
Tracción: [TRASERA]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Como habrás podido observar en el ejemplo anterior, se utiliza **this** para hacer referencia a Miembros de la propia clase.

MATERIAL Y/O EQUIPO A UTILIZAR:

- ☐ Computadora personal
- ☐ Programa JAVA Netbeans instalado
- ☐ Cuaderno de clases, donde están los ejercicios resueltos en clase

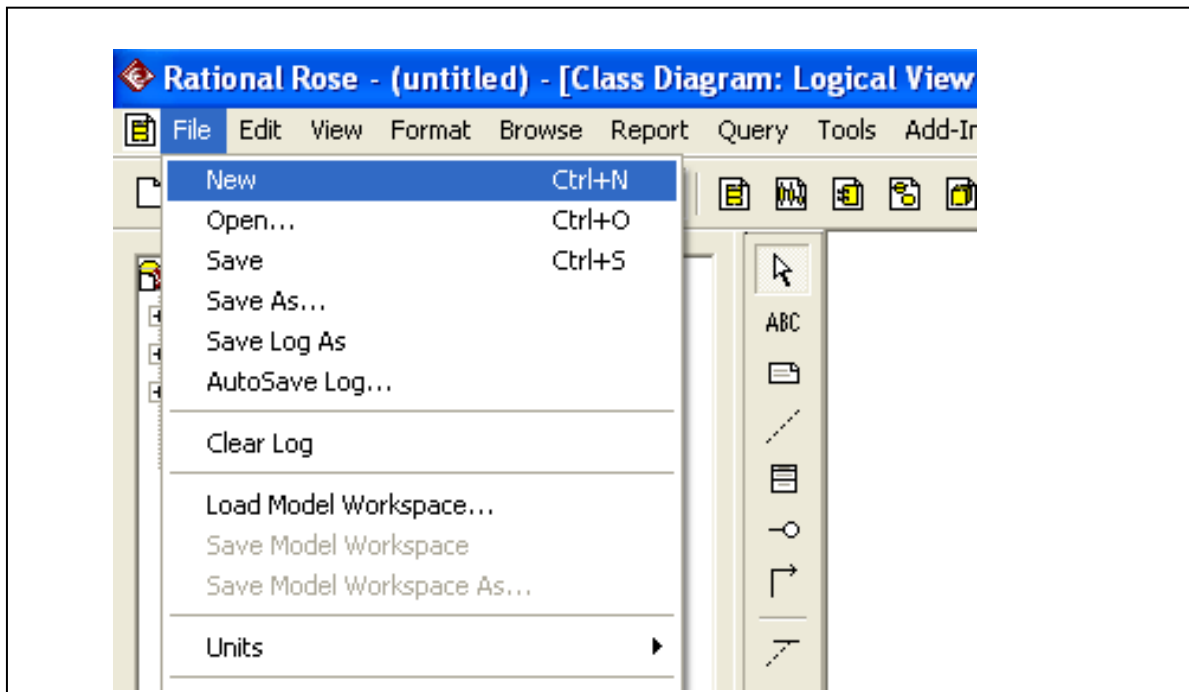
Diseñar un diagrama de clases en el lenguaje **UML**, que este en función al siguiente código.

```
public class Persona {
    private String nombre;
    private String apellido;
    private String sexo;
    private int edad;

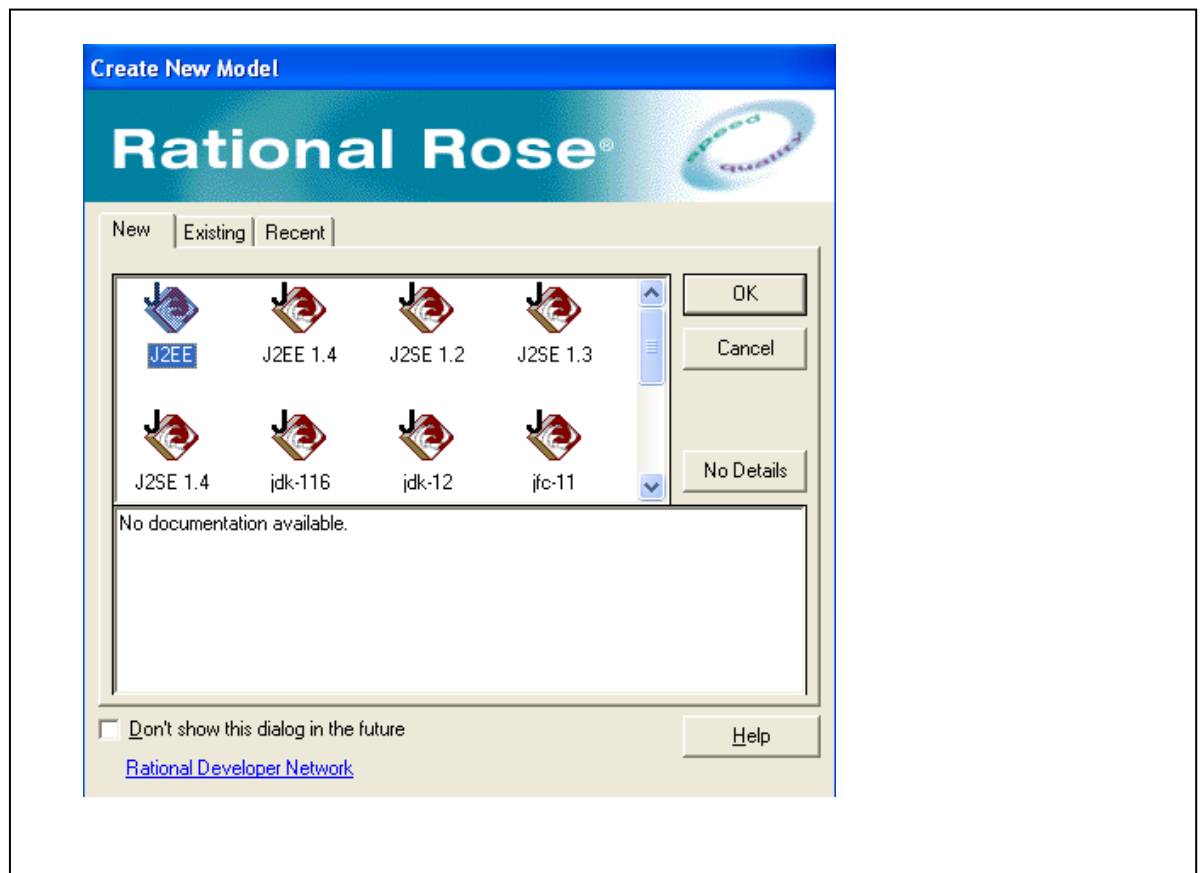
    public static void main(String[] args)
    {
    }
    public void Dormir()
    {
    }
    public void Bailar()
    {
    }
    public String HablarSutilmente(String mensaje)
    { String msj="";
      return msj;
    }
}
```

Solución:

- a) Para el respectivo diseño se puede utilizar el programa rational rose o sino el mismo modelador del netbeans en **UML**.
- b) Si utilizamos el rational rose versión 7.0 podemos crear un proyecto que me permita definir un diagrama de clases.
- c) Nosotros a continuación vamos a especificar un conjunto de pasos para poder crear todo lo que se ha especificado anteriormente.
- d) Para poder crear un proyecto en el rational rose, primeramente tenemos que ir al menú **File/new Project**.

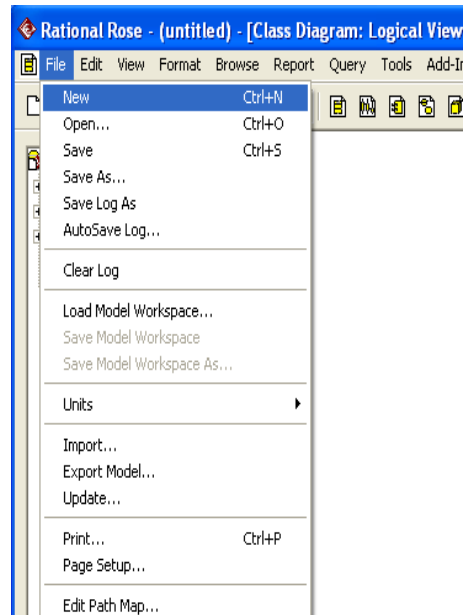


e) Luego aparece una ventana, donde se tendrá que presionar el botón cancelar

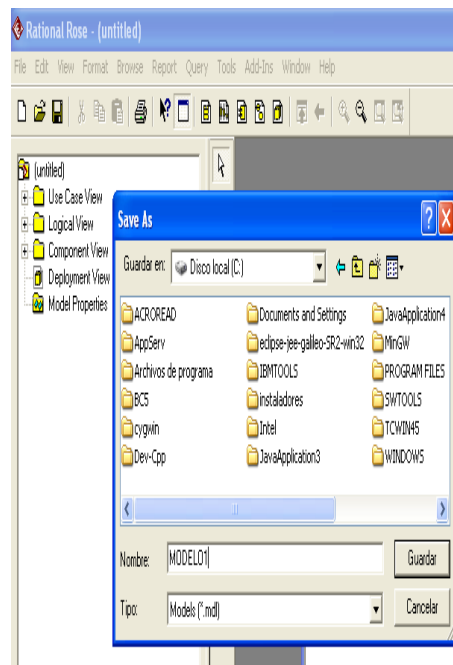


f) A continuación, ir al menú File y seleccionar new

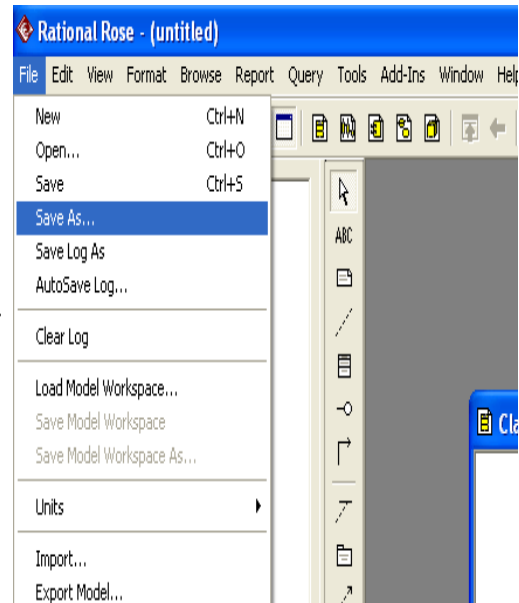
Paso # 01



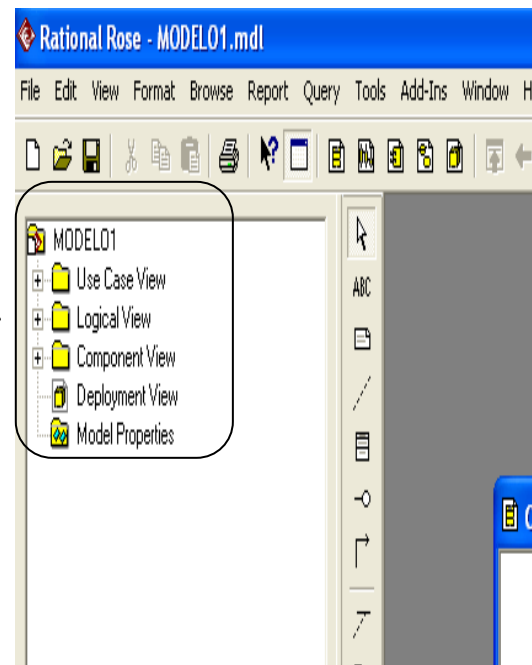
Paso # 03



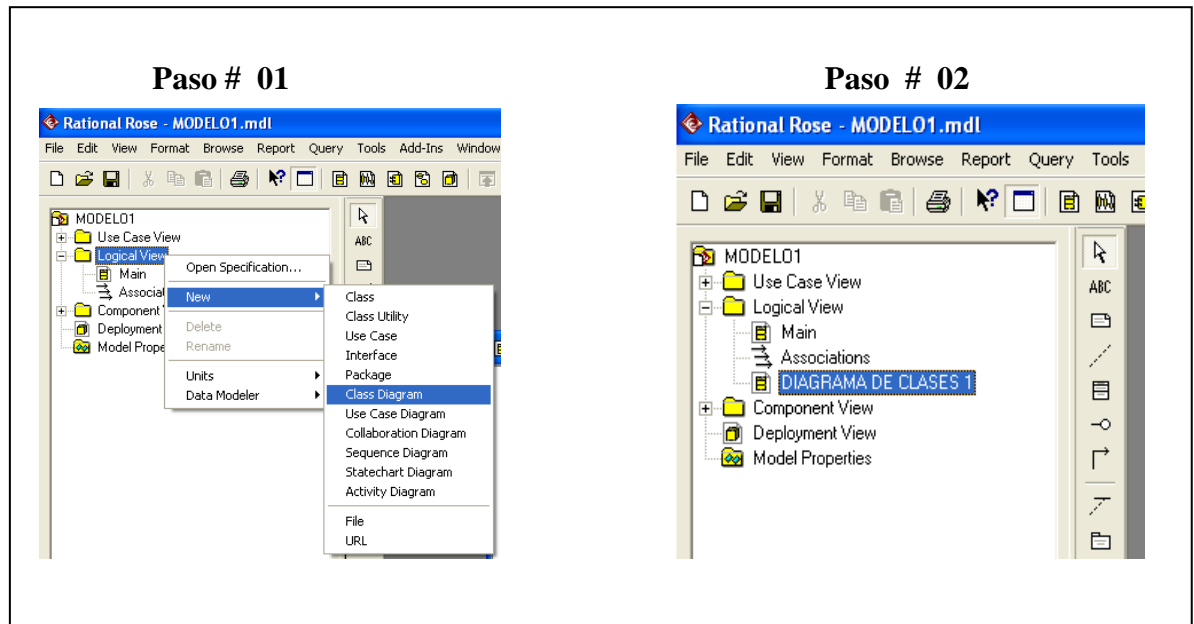
Paso # 02



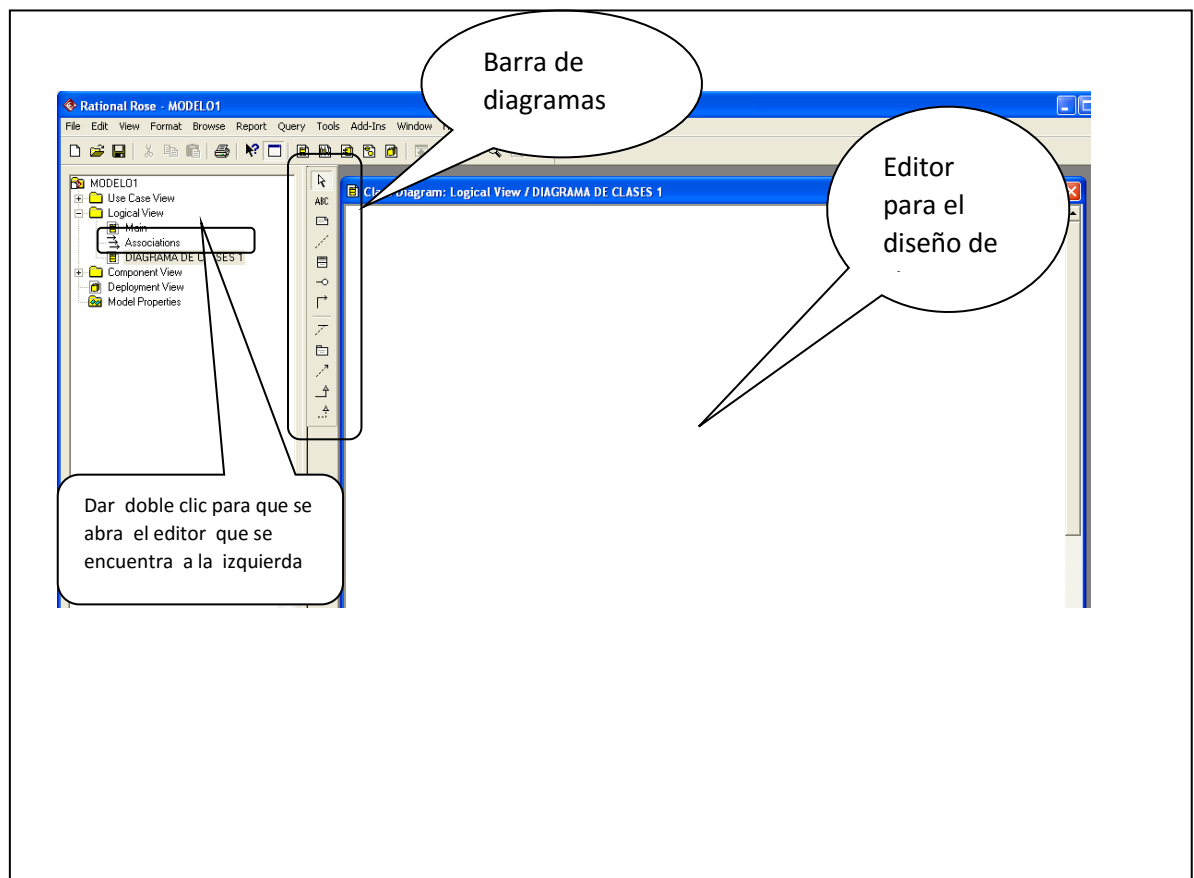
Paso # 04



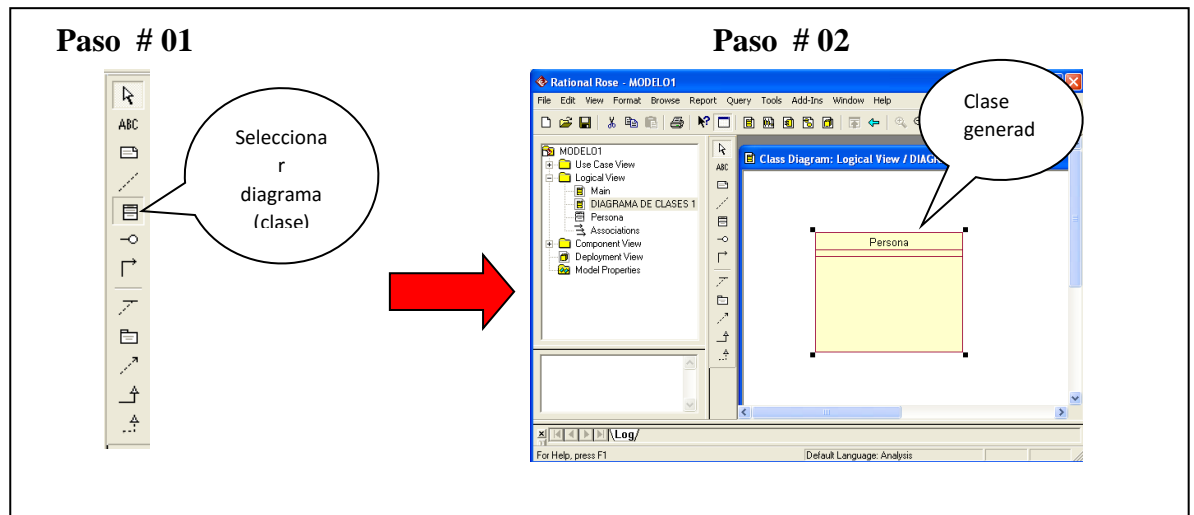
g) Ahora vamos a crear un diagrama de clases, dentro del paquete Logical view.



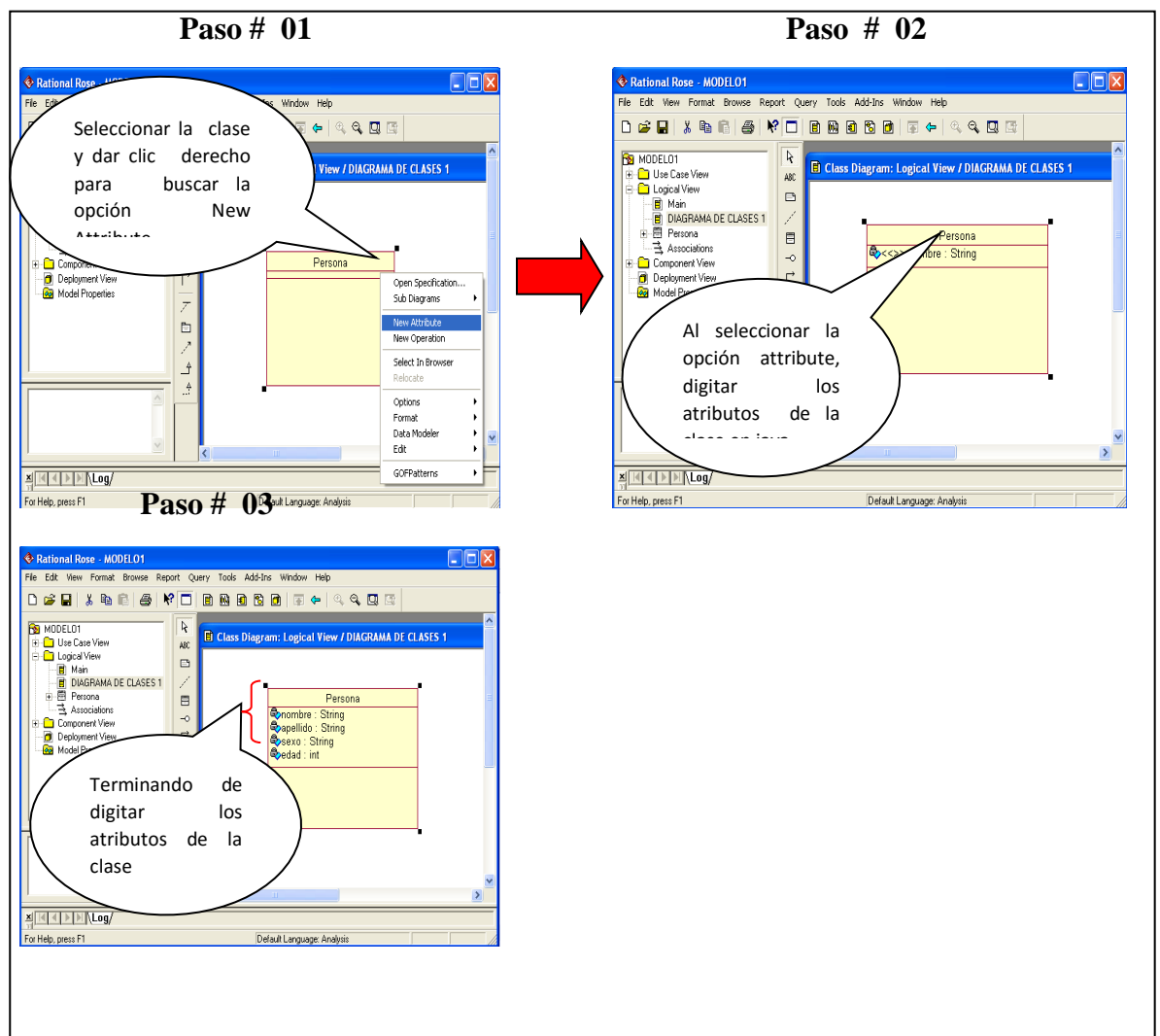
h) Creado el diagrama de clases, dar doble clic sobre este mismo para que se abra un editor ubicado a la izquierda.



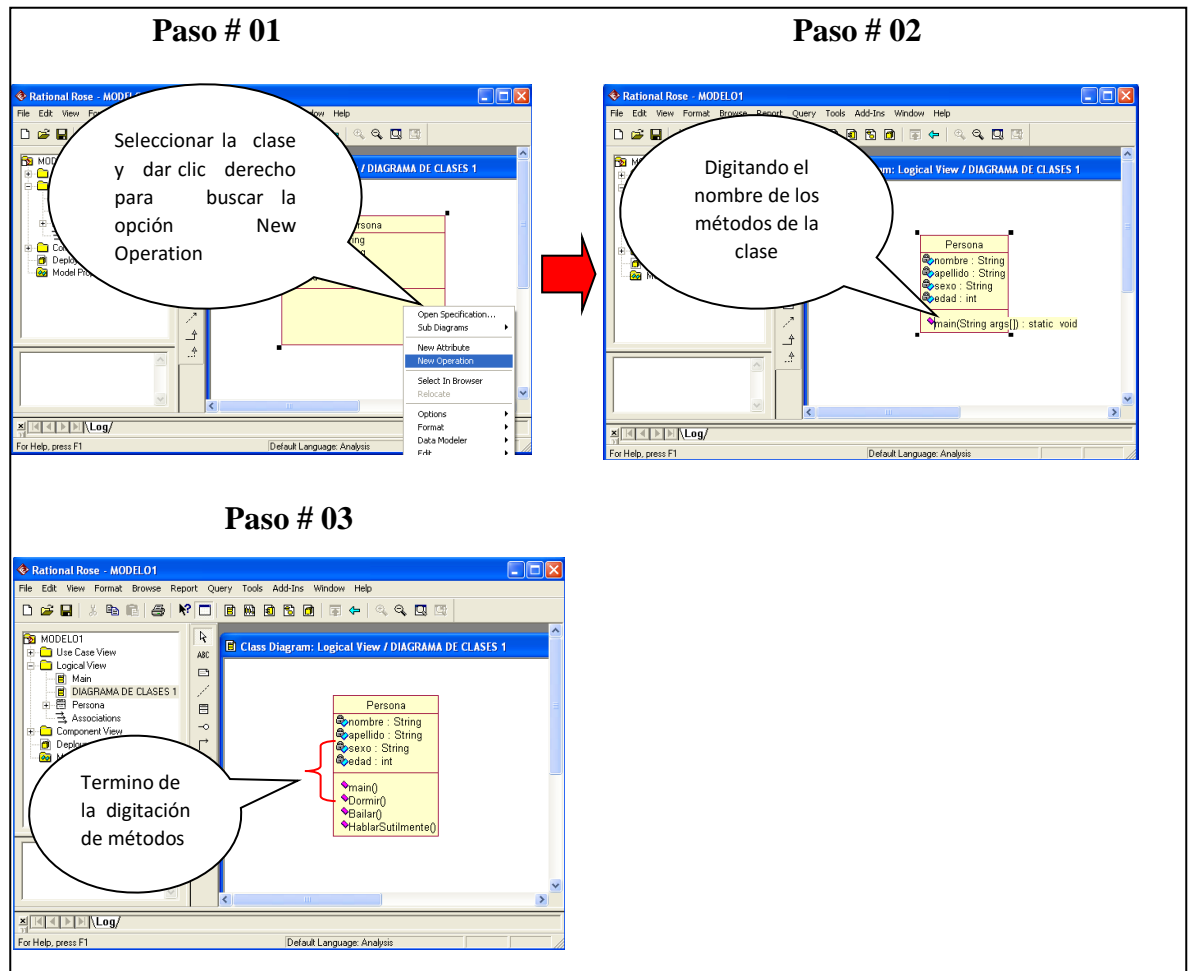
- i) Seleccionar una clase desde la barra de diagramas y arrastrar al editor de diseño, generando una clase.



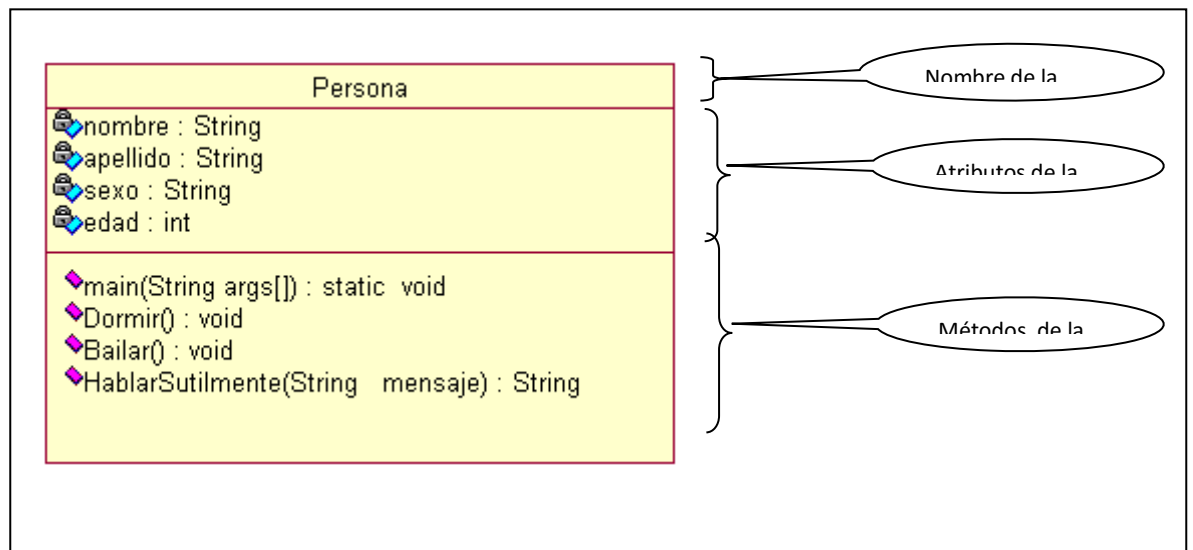
- j) Agregando los atributos a la clase.



k) Agregando los métodos a la clase

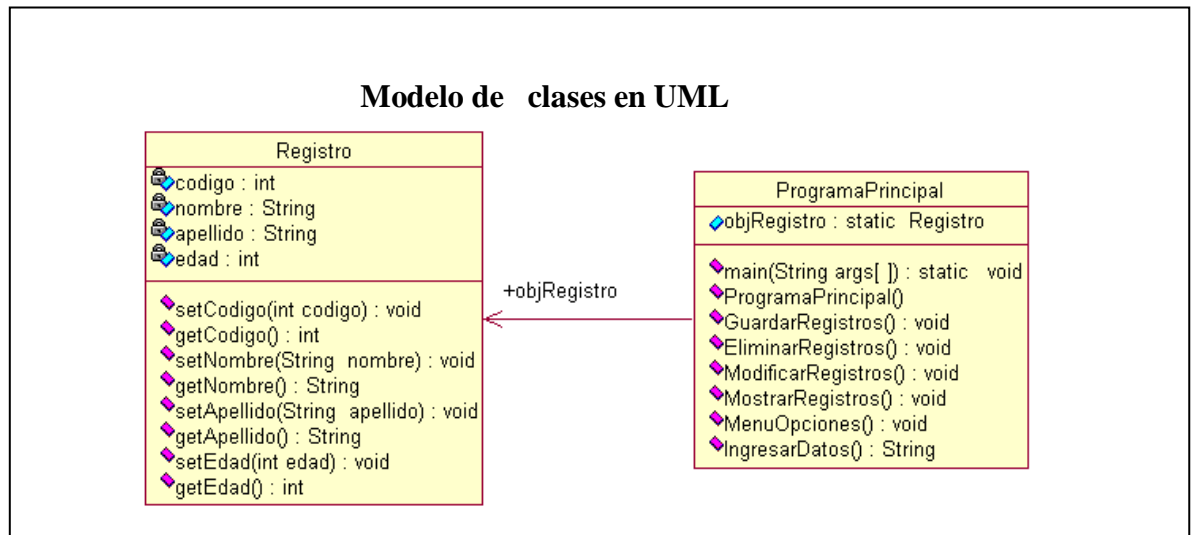


l) Finalmente este es el una clase en **UML** que nosotros debemos de diseñar.



EJERCICIOS # 02

El siguiente modelo representa un conjunto de clases en UML, traducir al lenguaje de programación Java el siguiente modelo.



Solución:

- a) Primeramente vamos a codificar la clase Registro en función al modelo en UML :

```
public class Registro
{
    private int    codigo ;
    private String nombre ;
    private String apellido ;
    private int    edad ;

    public void setCodigo(int codigo)
    {
        this.codigo = codigo;
    }
    public int getCodigo()
    {
        return codigo;
    }
}
```

```
public void setNombre(String nombre)
{
    this.nombre = nombre;
}
public String getNombre()
{
    return nombre;
}
public void setApellido(String apellido)
{
    this.apellido = apellido;
}
public String getApellido()
{
    return apellido;
}
public void setEdad(int edad)
{
    this.edad = edad;
}
public int getEdad()
{
    return edad;
}
}
```

b) Codificando la clase **ProgramaPrincipal** :

```
public class ProgramaPrincipal {  
  
    public static Registro objRegistro;  
  
    public static void main(String[] args)  
    {  
        objRegistro=new Registro();  
    }  
    public ProgramaPrincipal()  
    {  
    }  
    public void GuardarRegistros()  
    {  
    }  
    public void EliminarRegistros()  
    {  
    }  
    public void ModificarRegistros()  
    public void MostrarRegistros()  
    {  
    }  
    public void MenuOpciones()  
    {  
    }  
    public void IngresarDatos()  
    {  
    }  
}
```

EJERCICIO #03

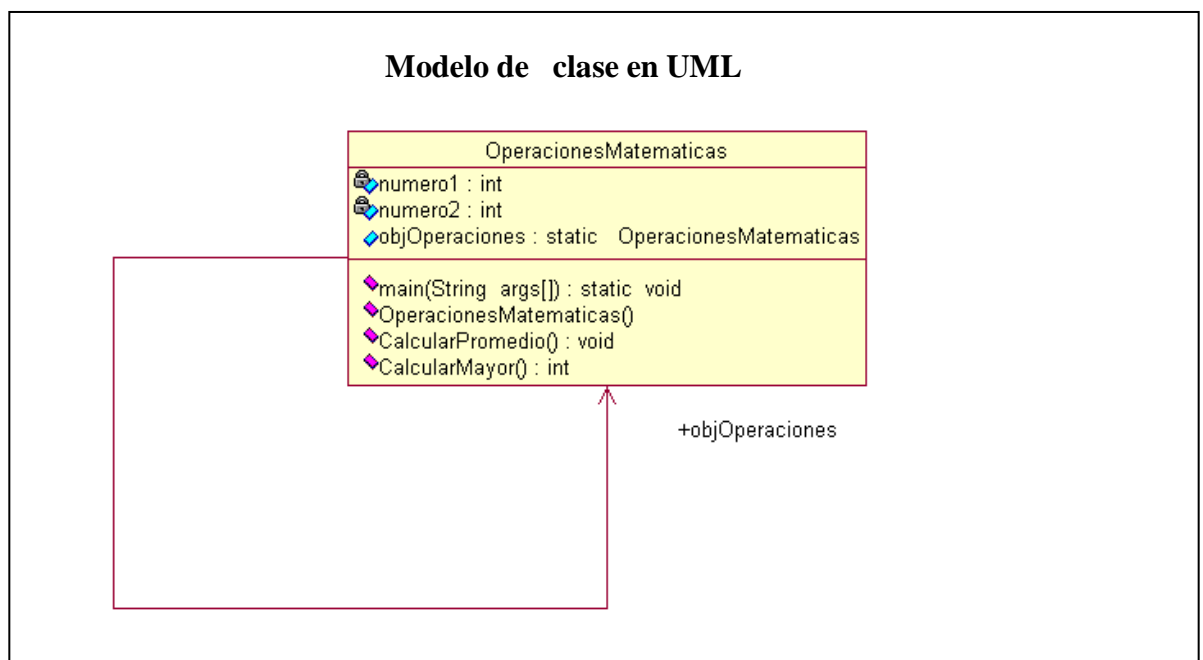
El siguiente modelo representa una clase en **UML de nombre OperacionesMatematicas**, que tiene como atributos , dos números enteros , cuyo modificador de acceso es privado , una referencia de la misma clase de modificador de acceso público , que me permite apuntar a un objeto de la misma clase.

Esta referencia que apunta al objeto me permite la invocación de todos los métodos implementados en la clase.

La clase **OperacionesMatematicas**, tiene implementado dos métodos , la primera calcula el promedio y no retorna valor , el segundo calcula el mayor y retorna valor

Ambos métodos tiene un modificador de acceso público y para su respectivo cálculo, utilizan los atributos numéricos publicados en la clase.

Además la clase **OperacionesMatematicas** tiene implementado un constructor para su respectiva inicialización de los atributos numéricos de la clase.



Desarrollar un programa en java que me permita contemplar los requerimientos visto anteriormente y la estructura de la clase representado en **UML**.

Solución:

- a) Primeramente tenemos que leer cuidadosamente los requerimientos solicitados en el enunciado del problema , además respetar las arquitectura que nos muestra en el modelo de clase en **UML**.
- b) A continuación vamos a codificar en Java la arquitectura de clase que se especifica en el modelo **UML**.

```
public class OperacionesMatematicas
{
    private int numero1;

    private int numero2;

    public static OperacionesMatematicas objOperaciones ;

    public static void main(String[] args)
    {
        objOperaciones=new OperacionesMatematicas();
    }

    public OperacionesMatematicas()
    {
    }

    public void CalcularPromedio()
    {

    }

    public int CalcularMayor()
    {
        int mayor=0;

        return mayor;
    }
}
```

- c) Codificado la clase en el lenguaje de programación Java, vamos a implementar el contenidos de los métodos.

```
public class OperacionesMatematicas
{
    private int numero1;

    private int numero2;

    public static OperacionesMatematicas objOperaciones ;

    public static void main(String[] args)
    {
        objOperaciones=new OperacionesMatematicas();

        objOperaciones.CalcularPromedio();

        System.out.println("EL MAYOR ES :"+objOperaciones.CalcularMayor());
    }

    public OperacionesMatematicas()
    {
        numero1=15 ;
    }
    public void CalcularPromedio()
    {
        double promedio =0.0 ;

        promedio= (double) (numero1+numero2)/2;

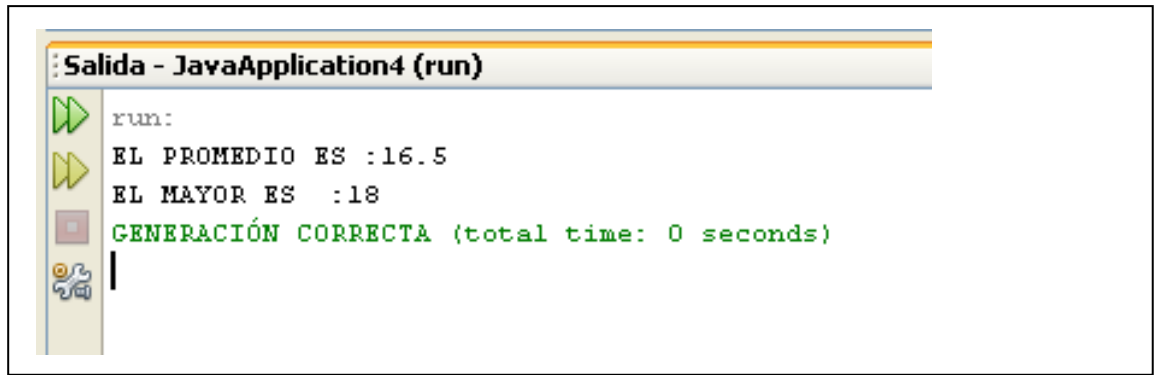
        System.out.println("EL PROMEDIO ES :"+promedio);
    }

    public int CalcularMayor()
    {
        int mayor=0;

        mayor= Math.max(numero1,numero2);

        return mayor;
    }
}
```

d) Finalmente ejecutamos el programa.



```
Salida - JavaApplication4 (run)
run:
EL PROMEDIO ES :16.5
EL MAYOR ES :18
GENERACIÓN CORRECTA (total time: 0 seconds)
```

Guía Práctica 1

1. Diseñar un programa que calcule el monto que se pagará por la compra de ciertas unidades de un producto.
2. Pedro, Juan y María aportan cantidades de dinero para formar un capital, Pedro y María aportan en dólares y Juan en soles. Diseñe un programa que determine el capital total en dólares y calcule que porcentaje de dicho capital aporta cada uno. Considere que 1 dólar = 2.79 soles.
3. Diseñe un programa que determine el área y el perímetro de un rectángulo, sabiendo que las fórmulas son:
Area = base * altura
Perímetro = 2 * (base + altura)
4. Diseñe un programa para repartir una cantidad de dinero a cuatro personas en forma proporcional a sus edades. El monto que le corresponde a cada persona se calcula con la siguiente formula:
$$\text{Monto de la persona} = \frac{\text{Edad de la persona} * \text{monto a repartir}}{\text{Suma total de edades}}$$
5. En una tienda se ha puesto en oferta la venta de cierto tipo de producto ofreciendo un descuento fijo del 11% del monto de la compra. Diseñe un programa que determine el monto del descuento y el monto total a pagar.
6. Un vendedor de terrenos ofrece a 120 soles el metro cuadrado, además cobra 80 soles por instalación de luz y 75 por agua potable. El programa debe calcular el costo final de dicho terreno.

Bibliografía:

- THOMAS WU C. Introducción a la programación orientada a objetos con Java. 1ª Edición. España. McGraw-Hill Interamericana de España. 2008. 15-22pp. ISBN: 978-0-07-352339-2
- LEOBARDO LOPEZ. Román. Metodología de la programación orientada a objetos. 1ª Edición. México. Alafomega grupo editor de México. 2006. 241-253pp ISBN: 970-15-1173-5
- HERBERT SHILDT. JAVA 2 v5.0. España. Ediciones Anaya multimedia.2005. 79-99pp ISBN: 84-415-1865-3

REFERENCIAS EN INTERNET

- <http://gcoronelc.blogspot.pe>
- <http://gcoronelc.blogspot.pe/2016/11/programando-pensando-en-servicios-parte.html>
- <http://gcoronelc.blogspot.pe/2016/11/prog-pensando-en-servicios-parte-2.html>
- <http://gcoronelc.blogspot.pe/2016/06/separata-java-orientado-objetos.html>
- <http://gcoronelc.blogspot.pe/p/java.html>
- <http://gcoronelc.blogspot.pe/2017/01/java-fundamentos-01-introduccion.html>
- <http://gcoronelc.blogspot.pe/2013/09/java-poo-leccion-01.html>
- <http://gcoronelc.blogspot.pe/2013/09/java-poo-leccion-02.html>