

TRABAJO PRACTICO N° 4

PARADIGMAS DE PROGRAMACIÓN

Dr. Pablo Javier Vidal






Unidad 2

Relaciones

Ejercicio 1.

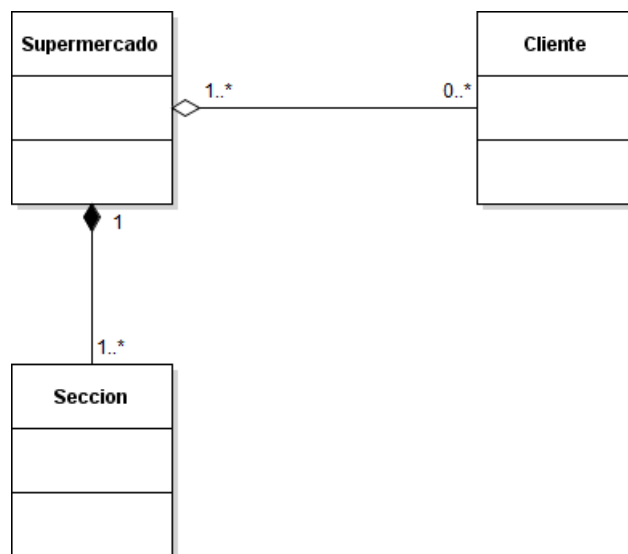
Considere la siguiente tabla de términos. La primera columna contiene tipos de relaciones que se pueden dar entre clases. La columna del centro, tiene los términos utilizados frecuentemente cuando se leen relaciones entre clases a partir de un diagrama. La columna de la derecha se ven las representaciones gráficas en los diagramas.

Indicar los conjuntos que tienen relación entre si utilizando las letras de cada columna, ejemplo: a-c-d

Tipo de Relación	Termino usado cuando se leen las relaciones entre clases	Representación en diagrama
a)Especialización/Generalización (Herencia)	a)"Es un tipo de"	a) 
b)Dependencia	b)"Es parte de"	b) 
c)Agregación	c)"Está formado por"	c) 
d)Composición	d)"Depende de"	d) 
e)Asociación	e)"Se conecta a" ó "Se asocia a"	e) 

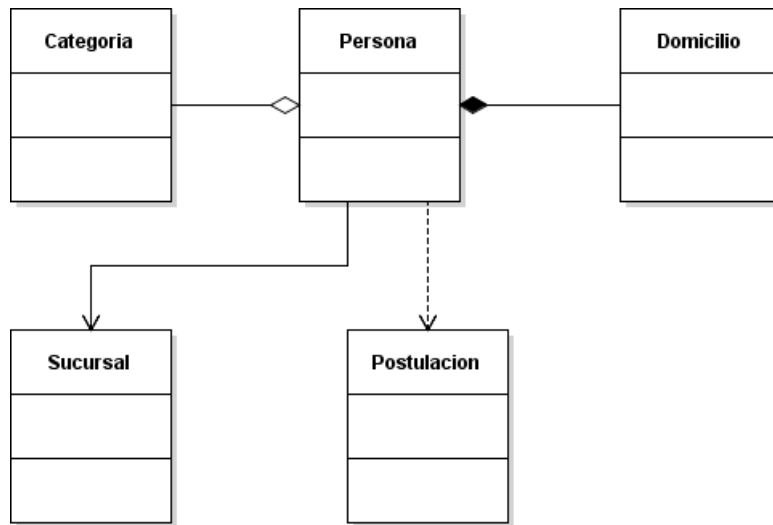
Ejercicio 2.

Para el siguiente diagrama mencione el o los tipos de relaciones existentes y escriba una interpretación al lenguaje natural.



Ejercicio 3.

Para el siguiente diagrama mencione el o los tipos de relaciones existentes, las cardinalidades e implemente todas las relaciones en lenguaje Java. Defina al menos dos atributos por cada clase y al menos dos métodos relacionados con la clase.



Ejercicio 4.

Definir la clase *Fecha*, la cual debe tener un constructor que reciba tres valores enteros representando el día, mes y año que serán usados para instanciar las clases *Dia*, *Mes*, *Anio*. Cada clase, *Dia*, *Mes* y *Anio* debe corroborar que el dato ingresado sea correcto. Asimismo la clase *Fecha* debe tener un método que informe el *siguienteDia()* y la clase *Año* que tenga el método para informar si es bisiesto o no. La clase *Fecha* debe tener las interfaces definidas (métodos) para poder mostrar el mensaje de los dos métodos anteriores.

Ejercicio 5.

Definir la clase asociación *ContratoCliente* entre las clases *Cuenta* y *Cliente*. Defina los atributos y métodos necesarios para cada clase incluyendo constructores, getters y setters.

Ejercicio 6.

Una *Persona* esta casada con otra *Persona*. Definir la relación existente.

Ejercicio 7.

Definir la clase *Polígono* la cual puede guardar objetos de tipo *Punto* para formar cada tipo de *Polígono*. Defina la relación existente entre las clases.

Ejercicio 8.

Crear una clase *Persona* con los siguientes atributos: nombre, dirección, códigoPostal, ciudad y fechaNacimiento. El atributo fechaNacimiento es un objeto de la clase *Fecha* creada en esta sección anteriormente. ¿Que tipo de relación entre clases existe?

Ejercicio 9.

Dado los siguientes casos:

1. Un país tiene provincias, limita con otros países y se localiza en un continente. Las provincias de un país limitan con otras provincias del mismo país. Las provincias tienen ciudades y una de ellas es su capital.
2. Un cliente puede realizar muchos pedidos. Cada pedido tiene una bebida que viene de una bodega que almacena diferentes bebidas. Cada bebida puede ser jugo natural, agua o gaseosa. Cada vez que un cliente hace un pedido a un mozo, se registra la venta de dicho pedido asociado a un mozo.

Proceder a realizar los siguientes pasos:

- Identifique clases y relaciones entre clases, clasificando estas en generalización/especialización, asociación, agregación o composición.
- Construya los diagramas de clases correspondientes. Incluya atributos y operaciones (métodos) si se requiere.

Ejercicio 10.

Se necesita implementar un sistema en el que se puedan cargar alumnos, a los cuales los caracterizan el nombre y apellido, el número de legajo, el sexo, condición (regular o condicional) y la nota final. Estos alumnos se deben cargar en una asignatura, llamada Algoritmos y Programación III.

Implemente las clases y métodos necesarios para esta situación, teniendo en cuenta lo que se pide a continuación:

1. Mostrar en pantalla todos los alumnos que se encuentren en la asignatura.
2. Mostrar en pantalla los alumnos que se encuentren como condicional y su cantidad.
3. Ordenar los alumnos de acuerdo a su nota (de mayor a menor) y mostrarlo en pantalla.
4. Ordenar los alumnos de acuerdo a su nota (de menor a mayor) y mostrarlo en pantalla.
5. Ordenar los alumnos por nombre y apellido y mostrarlo en pantalla

Nota: para los ordenamientos utilizar las facilidades provistas por la plataforma Java.

Herencia de Clases

Ejercicio 11.

Definir una jerarquía de herencia en la cual podamos especializar la clase Profesor en Profesor Efectivo y Profesor Interino. Cada profesor se diferencia en si el cargo que están ocupando ha sido obtenido mediante concurso o por resolución respectivamente. Agregar los getters y setters necesarios en cada clase, Agregar al menos un método en cada subclase profesor que acceda a los valores de la clase padre.

Ejercicio 12.

Construir una clase FacturaA, FacturaB y FacturaC que descienda de la clase Factura. Para la clase abstracta debe definir al menos dos atributos específicos llamados *emisor* y *cliente* y, al menos, un método llamado *imprimirFactura*. Para cada clase defina los atributos necesarios y métodos propios de cada especialización.

Ejercicio 13.

Construir una clase final *Math2* que amplíe las declaraciones de métodos estáticos de la clase *Math* y que incluya funciones que devuelvan, respectivamente, el máximo, el mínimo, el sumatorio, la media aritmética y la media geométrica de un array de números reales dado como parámetro.

Ejercicio 14.

Define tres clases: *Casa*, *SalonCasa* y *CocinaCasa*. La clase *SalonCasa* debe tener como atributos *numeroDeTelevisores* (int) y *tipoSalon* (String) y disponer de un constructor que los inicialice a 0 y “desconocido”.

La clase *Casa* tendrá los siguientes atributos de clase: *superficie* (double), *direccion* (String), *salonCasa* (tipo *SalonCasa*) y *cocina* (tipo *CocinaCasa*).

La clase *CocinaCasa* debe tener como atributos *esIndependiente* (boolean) y *numeroDeFuegos* (int) y un constructor que los inicialice a false y 0.

Definir un constructor para la clase *Casa* que establezca valores de defecto a los atributos simples y que cree nuevos objetos si se trata de atributos objeto.

Compilar el código para comprobar que no presenta errores, crear un objeto de tipo *Casa*. Comprueba que se inicializan correctamente consultando el valor de sus atributos después de haber creado los objetos.

Ejercicio 15.

La empresa informática *DuDu* necesita llevar un registro de todos sus empleados que se encuentran en la oficina central, para eso ha creado un diagrama de clases que debe incluir lo siguiente: Empleado:

- Atributos:
 - nombre: tipo cadena (Debe ser nombre y apellido)
 - cedula: tipo cadena
 - edad : entero (Rango entre 18 y 45 años)
 - casado: boolean
 - salario: tipo numérico doble
- Métodos:
 - Constructor con y sin parámetros de entrada
 - Método que permita mostrar la clasificación según la edad de acuerdo al siguiente algoritmo:
 - Si edad es menor o igual a 21, Principiante
 - Si edad es ≥ 22 y ≤ 35 , Intermedio
 - Si edad es > 35 , Senior.-
 - Imprimir los datos del empleado por pantalla (utilizar salto de línea
 para separar los atributos.
 - Un método que permita aumentar el salario en un porcentaje que sería pasado como parámetro al método.

Programador (Especialización de Empleado). Clase que hereda de Empleado todos los atributos y métodos.

- Atributos:

- lineasDeCodigoPorHora : tipo entero
 - lenguajeDominante: tipo cadena
- Metodos:
 - Constructor con y sin parámetros de entrada.

Ejercicio 16.

Un motor es la parte principal de una máquina capaz de hacer funcionar cualquier sistema, transformando algún tipo de energía (eléctrica, de combustibles fósiles, etc.), en energía mecánica capaz de realizar un trabajo.

El motor eléctrico es un dispositivo que convierte la energía eléctrica en energía mecánica rotación por medio de la acción de los campos magnéticos generados en sus bobinas. Son máquinas eléctricas que son impulsadas por fuentes de corriente continua (C.C.) y fuentes de corriente alterna (C.A.).

Un motor térmico es una máquina térmica que transforma calor en trabajo mecánico por medio del aprovechamiento del gradiente de temperatura entre una fuente de calor (foco caliente) y un sumidero de calor (foco frío). Los motores pueden tener un movimiento alternativo o rotativo lo cual supone un funcionamiento totalmente diferente.

Un motor de explosión es un tipo de motor de combustión interna que utiliza la explosión de un combustible, encendido de manera provocada mediante una chispa, para expandir un gas que empuja un pistón, el cual esta sujeto al cigüeñal por una biela, esta hace las veces de manivela y transforma el movimiento lineal del pistón en rotativo en el cigüeñal. Los motores de combustible pueden funcionar con combustible diésel o gasolina teniendo este ultimo motores de explosión de dos y de cuatro tiempos.

Definir el diagrama de clases indicando las relaciones de herencia y los atributos que irían en cada clase.

Interfaces

Ejercicio 17.

Construir una clase ArrayReales que declare un atributo de tipo double[] y que implemente una interfaz llamada Estadisticas. El contenido de esta interfaz es el siguiente:

```

1 public interface Estadisticas {
2     double minimo();
3     double maximo();
4     double sumatorio();
5 }
6

```

Ejercicio 18.

Construir una interfaz Relaciones (y posteriormente una clase que la implemente) que incluya los siguientes métodos:

```

1 // Devuelve verdadero si a es mayor que b
2 boolean esMayor(Object b) ;
3 // Devuelve verdadero si a es menor que b
4 boolean esMenor(Object b) ;
5 // Devuelve verdadero si a es igual que b
6 boolean esIgual(Object b) ;
7

```

Ejercicio 19.

Se plantea desarrollar un programa Java que permita representar la siguiente situación. Una instalación deportiva es un recinto delimitado donde se practican deportes, en Java interesa disponer de un método `int getTipoDeInstalacion()`. Un edificio es una construcción cubierta y en Java interesa disponer de un método `double getSuperficieEdificio()`. Un polideportivo es al mismo tiempo una instalación deportiva y un edificio; en Java interesa conocer la superficie que tiene y el nombre que tiene. Un edificio de oficinas es un edificio; en Java interesa conocer el número de oficinas que tiene.

Definir las interfaces y las clases necesarias para representar la situación anterior. En una clase `Test` con el método `main`, crear un `ArrayList` que contenga tres polideportivos y otro `ArrayList` que contenga dos edificios de oficinas y utilizando un `iterator`, recorrer la colección y mostrar los atributos de cada elemento. ¿Entre qué clases existe una relación que se asemeja a la herencia múltiple?

Polimorfismo

Ejercicio 20.

Tenemos nuestra clase padre *Animal* junto con sus 3 clases hijas *Perro*, *Gato*, *Caballo*. Crear un método abstracto en la clase *Animal* el cual cada clase hija deba mostrar un mensaje por pantalla informando de que se alimenta.

Generar una clase *Main* que realice lo siguiente:

```
1
2
3 public class TestRunner {
4
5     public static void main(String[] args) {
6
7         //-->Declaracion del objeto PERRO
8         Animal perro = new Perro("Stich", "Carnivoro", 15, "Doberman");
9         perro.Alimentarse();
10        //-->Declaracion de otro objeto PERRO
11        Perro perro1 = new Perro("Teddy", "Croquetas", 10, "Chihuahua");
12        perro1.Alimentarse();
13
14
15        //-->Declaracion del objeto Gato
16        Animal gato = new Gato("Pelusa", "Galletas", 15, "Siames");
17        gato.Alimentarse();
18        //-->Declaracion del objeto Caballo
19        Animal caballo = new Caballo("Spark", "Pasto", 25, "Fino");
20        caballo.Alimentarse();
21
22    }
23 }
```

Ejercicio 21.

Se tiene una clase *Instrumento*, la cual puede tener como métodos `tocar` (muestra que el instrumento está tocando), `tipo de instrumento` que es, y un método `afinar`. Definir la clase *Guitarra*, *Piano*, *Saxofon*, *Guzla*, *Charango*, *Ukelele*, todas ellas heredan de *instrumento*. Por otro lado tenemos la clase *Musica* que tendrá tres métodos:

- `afinar`: recibe una clase de tipo *Instrumento* y activa el método `afinar` y luego el `tocar`.
- `afinarTodo`: afina todos los instrumentos. Recibe un arreglo de instancias de clase *Instrumento* (`Instrumento []`).

- `main`: el `main` evalúa los métodos anteriormente definidos.

```

1
2  public static void main(String[] args) {
3      Instrumento[] orquesta = new Instrumento[6];
4      int i = 0;
5      // Up-casting al asignarse el Arreglo
6      orquesta[i++] = new Guitarra();
7      orquesta[i++] = new Piano();
8      orquesta[i++] = new Saxofon();
9      orquesta[i++] = new Guzla();
10     orquesta[i++] = new Charango();
11     orquesta[i++] = new Ukelele();
12     afinarTodo(orquesta);
13 }
14

```

Nota: los métodos `afinar` y `afinarTodo` pueden ser *static* si así lo desea.

Genericidad

Ejercicio 22.

Definir la clase `Contenedor<T>` con un atributo contenido de tipo `T`. Definir los métodos `getters` y `setters` para el atributo.

1. Crear una clase de prueba e implementar un método *main* que posibilite generar una instancia, setear el valor del atributo e imprimirlo por pantalla
2. Agregar otro atributo genérico de nombre valor.
3. Asignar un tipo primitivo *int* al generar una nueva instancia para el atributo valor.
4. Asignar un tipo `Integer` al generar una nueva instancia para el atributo valor.

Ejercicio 23.

Crear la clase *Persona* con su *nombre*, *altura*, *edad*. Definir una instancia de *ArrayList* de personas instanciadas y luego:

1. Definir la clase *Persona* de forma tal que implemente la interface *Comparable* (*implements Comparable<Persona>*)
2. Definir 2 clases para comparar personas (una por edad y otra por altura) que implementen *Comparator<Persona>*
3. Use el método `sort` de *ArrayList* pasándole cada comparador e imprima la lista ordenada en cada caso.

Ejercicio 24.

Definir la clase *PlazoFijo* que solo acepte tipos genéricos compatibles con la clase *Deposito*. Utilizar una *LinkedList* para guardar los depósitos ingresados.

Ejercicio 25.

Implementar una clase *Garaje*, que únicamente permita almacenar *Vehiculos*, utilizando un *ArrayList*.

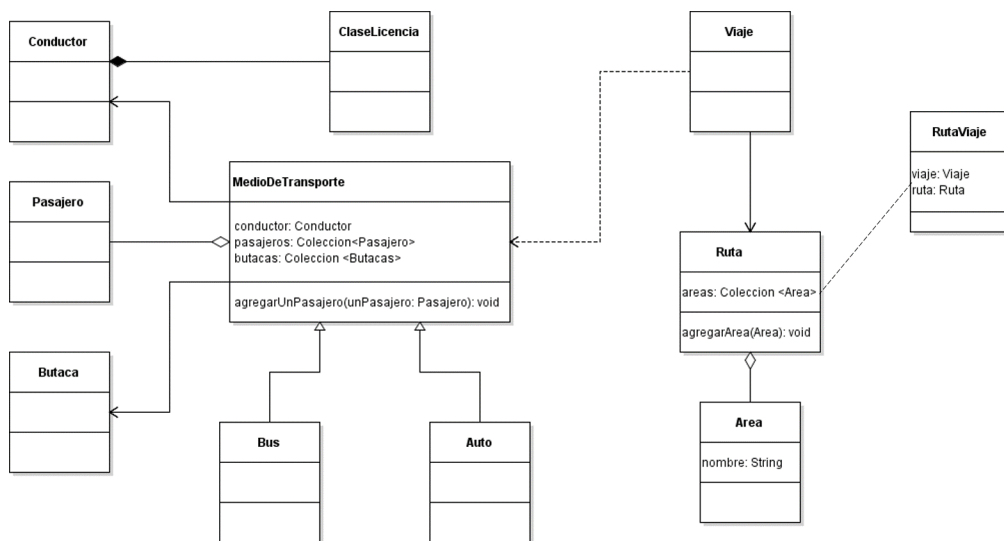
1. Métodos de la clase *Garaje*:

- estacionar: Recibe un vehículo y lo guarda en el array
- retirar: Elimina el vehículo del array. Lanza la excepción
- *VehiculoInexistente*, si no estaba en el garaje.

Ejercicio 26.

A continuación se muestra un diagrama de clases que representa una posible solución Se pide:

1. Identificar las relaciones existentes en el diagrama de clases.
2. Implementar el diagrama de clases con sus atributos, métodos y constructores para cada clase.
3. Agregar la interface *IFacturable* con el método *imprimirBoleta()* donde Ud. crea más conveniente.



Colecciones

Ejercicio 27.

Utilizando una lista (*ArrayList*) para almacenar 100 número aleatorios entre 1 y 100. Usar un *Iterator* para mostrar todos los números. Por último, mostrar los elementos de la lista sin repetir y en orden ascendente. Nota: utilizar la clase *Integer*.

Ejercicio 28.

Diseñar un programa que lea una serie de valores numéricos enteros desde el teclado y los guarde en un *ArrayList* de tipo *Integer*. La lectura de números termina cuando se introduzca el valor -99. Este valor no se guarda en el *ArrayList*. A continuación el programa mostrará por

pantalla el número de valores que se han leído, su suma, su media. Por último se mostrarán todos los valores leídos, indicando cuántos de ellos son mayores que la media.

Vamos a utilizar 3 métodos además del método `main` para resolver el ejercicio:

1. Método `leerValores()`: pide por teclado los números y los almacena en el *ArrayList*. La lectura acaba cuando se introduce el valor -99. El método devuelve mediante `return` el *ArrayList* con los valores introducidos.
2. Método `calcularSuma()`: Recibe como parámetro el *ArrayList* con los valores numéricos y calcula y devuelve su suma. En este método se utiliza un *Iterator* para recorrer el *ArrayList*.
3. Método `mostrarResultados()`: Recibe como parámetro el *ArrayList*, la suma y la media aritmética. Muestra por pantalla todos los valores, su suma y su media y calcula y muestra cuantos números son superiores a la media. En este método se utiliza un `for` para colecciones para recorrer el *ArrayList*.

Ejercicio 29.

Crear una clase denominada *ListaCantantesFamosos* que disponga de un atributo *ArrayList listaCantantesFamosos* que contenga objetos de tipo *CantanteFamoso*. La clase debe tener un método que permita añadir objetos de tipo *CantanteFamoso* a la lista.

Un objeto de tipo *CantanteFamoso* tendrá como atributos *nombre* (*String*) y *discoConMasVentas* (*String*), y los métodos para obtener y establecer los atributos. Crea una clase test con el método `main` que inicialice un objeto *ListaCantantesFamosos* y añada manualmente dos objetos de tipo *CantanteFamoso* a la lista.

Usando *Iterator* muestra los nombres de cada cantante y su disco con más ventas por pantalla.

Se debe pedir al usuario un nombre y disco con más ventas de otro cantante famoso, y una vez introducidos los datos mostrar la lista actualizada usando *Iterator*. Una vez mostrada la lista actualizada, se debe dar opción a elegir entre volver a introducir los datos de otro cantante o salir del programa (se podrán introducir tantos datos de cantantes como se desee. Para ello usa un bucle `while` que dé opción a elegir al usuario).

Marco Teórico

Ejercicio 30.

Responda y justifique cada una de las preguntas en caso de ser necesario:

1. ¿Cuál la clave en el proceso de análisis y diseño orientado a objetos, ya que mediante ella podemos llegar a armar un conjunto de clases?
2. ¿Cómo definiría al polimorfismo según sus palabras?
3. ¿Concepto que se define como la declaración de un objeto?
4. ¿Cuál es el pilar de la POO que consiste en unir en una clase las características y comportamientos abstraídos de un objeto?
5. ¿Encapsulamiento: Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción? ¿Por qué?
6. ¿Cuál es el pilar de la POO que consiste en captar las características, comportamientos e identidad que distinguen a un objeto?
7. ¿Cuál es el pilar de la POO que solamente aplica sobre los métodos y que se refiere a que el mismo método puede ser usado para diferentes fines según se necesite?

Ejercicio 31.

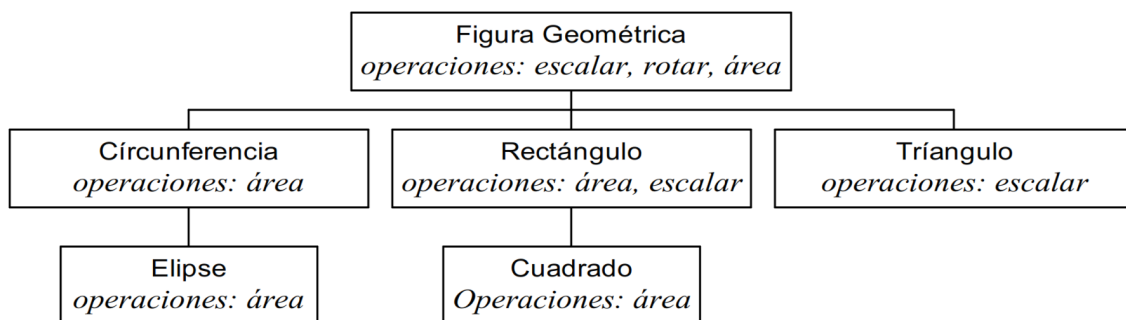
Describa cómo los lenguajes orientados a objetos PUROS brindan polimorfismo y son a la vez fuertemente tipados.

Ejercicio 32.

Muestre algún ejemplo en donde la clasificación con herencia múltiple permita un modelado más natural que la lograda con herencia simple. Investigue acerca de cuales son los lenguajes que soportan herencia múltiple. Analice bajo qué circunstancias la herencia múltiple puede presentar problemas. ¿Qué mecanismos pueden implementarse para evitarlos? Describir cada uno de ellos.

Ejercicio 33.

Analice la siguiente jerarquía de clases, asumiendo herencia simple:



De la clasificación anterior se aprecia que todas las instancias de las clases Figura Geométrica, Círculo, Elipse, Rectángulo, Cuadrado y Triángulo poseen en su protocolo las operaciones escalar, rotar y área. Dentro de cada clase se indica las operaciones definidas. Por ejemplo, escalar es definida en Figura Geométrica y redefinida en Rectángulo y Triángulo. Por otra parte, área esta definida en Figura Geométrica y redefinida en Círculo, Elipse, Rectángulo y Cuadrado. La pregunta es: ¿son polimorfas las operaciones escalar, rotar y área? Justifique sus afirmaciones.

Ejercicio 34.

Defina y diferencie (si existe) la herencia de clases con interfaces en el lenguaje Java. ¿Podríamos tener solo herencia de clases como forma de herencia múltiple?

Ejercicio 35.

1. Explique las diferencias entre sobreescritura (Overriding) y sobrecarga (overloading). ¿Que conceptos trabajan en cada uno de ellos o están relacionados?
2. Que concepto visto en clase nos ayuda a solucionar en parte trabajar con sobrecarga de métodos.