UNIVERSIDAD CATÓLICA DE SANTA MARÍA
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

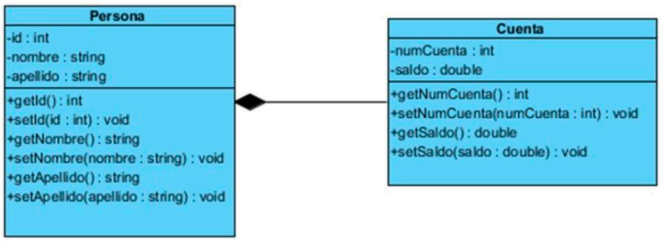
LENGUAJE DE LA PROGRAMACION III
RELACIONES ENTRE CLASES: AGREGACIÓN, COMPOSICIÓN Y HERENCIA

CÓDIGO/DNI	APELLIDOS Y NOMBRES	FECHA
2022701501	CAJA CCAPA ABDIEL JOAQUIN	10/09/2023
2022246652	MAMANI MENDIGURI MIRELLA THIANI	10/09/2023

ACTIVIDADES:

1.- Relación de composición:

a) Sea el siguiente diagrama de clases, en donde observamos la relación de composición entre la clase **Persona** y **Cuenta**:

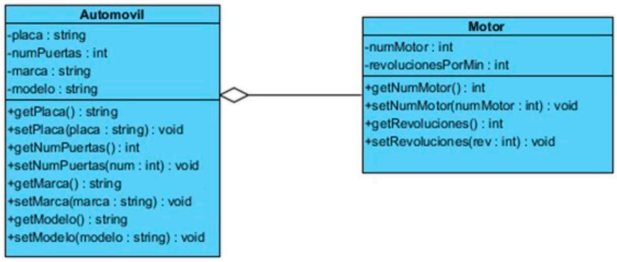


b) Cree la clase **TestComposicion** en donde se utilice objetos personas y muestre la información de las mismas.

```
2   import java.util.Scanner;
3   public class TestComposicion {
4       public static void main(String[] args) {
5           Cuenta cuenta1 = new Cuenta(2023, 200.0);
6
7           Persona persona1 = new Persona(1, "Jair", "Lopez", cuenta1);
8
9           System.out.println(persona1);
10
11          System.out.println("\n\nNúmero de cuenta: " + persona1.getCuenta().getNumero());
12          System.out.println("Saldo de la cuenta: " + persona1.getCuenta().getSaldo());
13      }
14  }
```

2.- Relación de agregación:

a) Sea el siguiente diagrama de clases, en donde observamos la relación de agregación entre la clase **Automovil** y **Motor**:



b) Implemente los métodos **getter** y **setter** de ambas clases.

```java
    public String getPlaca() {
        return placa;
    }

    public void setPlaca(String placa) {
        this.placa = placa;
    }

    public int getNumPuertas() {
        return numPuertas;
    }

    public void setNumPuertas(int numPuertas) {
        this.numPuertas = numPuertas;
    }

    public String getMarca() {
        return marca;
    }

    public void setMarca(String marca) {
        this.marca = marca;
    }

    public String getModelo() {
        return modelo;
    }

    public void setModelo(String modelo) {
        this.modelo = modelo;
    }

    public Motor getMotor() {
        return motor;
    }

    public void setMotor(Motor motor) {
        this.motor = motor;
    }
```

```java
    public int getNumMotor() {
        return numMotor;
    }

    public void setNumMotor(int numMotor) {
        this.numMotor = numMotor;
    }

    public int getRevPorMin() {
        return revPorMin;
    }

    public void setRevPorMin(int revPorMin) {
        this.revPorMin = revPorMin;
    }
```

c) Elmétodo **toString()** de ambas clases deben de permitir retornar la información de un automóvil.

```java
    @Override
    public String toString() {
        return "Automovil:" + "\nPlaca: " + placa + ",\nNúmero de puertas: " + numPuertas + "\nMarca:" + marca + "\nModelo:" + modelo + motor;
    }
```

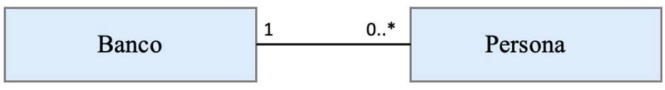
```java
    @Override
    public String toString() {
        return "\nMotor: " + "\nNúmero de motor: " + numMotor + "\nRevoluciones por minuto: " + revPorMin;
    }
```

d) Declare la clase **TestAgregacion**, y cree objetos **Automovil** y **Motor**. Asigne a los autormoviles creados un determinado motor, y muestre los datos de estos objetos.

```java
import java.util.Scanner;
public class TestAgregacion {

    public static void main(String[] args) {
        Motor motor1 = new Motor(25, 100);

        Automovil automovil1 = new Automovil("A85AC",4,"TOYOTA","Camry", motor1);

        System.out.println(automovil1);
    }
}
```

3.- Relación de asociasión:

a) Sea el siguiente diagrama de clases, en donde observamos la relación de asociación entre las clases **Banco** y **Persona** (definida en el ejercicio 1), la misma que representa el hecho que un banco tiene muchos clientes (personas), y una persona solo es cliente de un banco (esto último no necesariamente es así en un escenario real).



b) Completelaimplementacióndelasclasesteniendoencuentalosiguiente:

- Agregar el método buscarCliente de la clase Banco. Para esto debe utilizar el método equals().
- Agregue el metodo agregarCliente de la clase Banco. Se debe de verificar lla capacidad del arreglo para agregar al nuevo cliente. Además, si el cliente ya existe no debe ser duplicado.
- Implemente los métodos getter y setter.
- Implemente el método toString() que retorne la información de todos los clientes del banco.

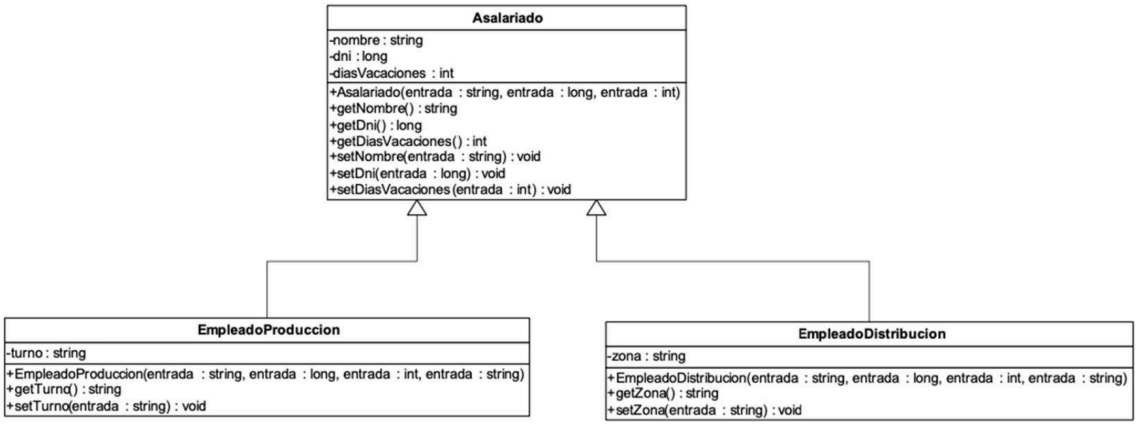
```java
    public String getNombre() {
        return nombre;
    }

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

    public Persona[] getClientes() {
        return clientes;
    }

    public void agregarCliente(Persona persona) {
        if (!buscarCliente(persona)) {
            if (cantidadClientes < clientes.length) {
                clientes[cantidadClientes] = persona;
                cantidadClientes++;
            } else {
                System.out.println("No hay espacio disponible para agregar más clientes.");
            }
        } else {
            System.out.println("El cliente ya existe en el banco y no puede ser duplicado.");
        }
    }

    public Persona darBajaCliente(Persona persona) {
        return null;
    }

    public boolean buscarCliente(Persona persona) {
        for (int i = 0; i < cantidadClientes; i++) {
            if (clientes[i].equals(persona)) {
                return true;
            }
        }
        return false;
    }
```

c) Cree la clase TestAsociacion y escriba las instrucciones que permitan agregar, buscar clientes y mostrar los clientes que tiene el banco.

```java
import java.util.Scanner;

public class TestAsociacion {

    public static void main(String[] args) {
        Banco bcp = new Banco("BCP");
        Banco bbva = new Banco("BBVA", 30);
        Scanner scanner = new Scanner(System.in);

        bcp.agregarCliente(new Persona("Juan Perez"));
        bcp.agregarCliente(new Persona("Maria Gomez"));

        bbva.agregarCliente(new Persona("Carlos Lopez"));

        System.out.println("Clientes del BCP:");
        for (Persona cliente : bcp.getClientes()) {
            if (cliente != null) {
                System.out.println(cliente.getNombre());
            }
        }

        System.out.println("Clientes del BBVA:");
        for (Persona cliente : bbva.getClientes()) {
            if (cliente != null) {
                System.out.println(cliente.getNombre());
            }
        }

        System.out.println("Ingrese el nombre del cliente a buscar en el BCP:");
        String nombreCliente = scanner.nextLine();
        boolean encontrado = bcp.buscarCliente(new Persona(nombreCliente));
        if (encontrado) {
            System.out.println("El cliente " + nombreCliente + " se encuentra en el BCP.");
        } else {
            System.out.println("El cliente " + nombreCliente + " no se encuentra en el BCP.");
        }

        scanner.close();
    }
}
```

4.- Relaciones de herencia:

a) Sea el siguiente diagrama de clases, en donde observamos la relación de herencia:



5.- Clases abstractas:

a) Escriba y analice las siguientes clases:

```
public abstract class FiguraGeometrica {
    private String nombre;

    abstract public double area();
    public figuraGeometrica (String nombreFigura ) {
        this.nombre = nombreFigura;
    }

    final public boolean mayorQue (FiguraGeometrica otra) {
        return this.area()>otra.area();
    }

    final public String toString()
        return this.nombre + " con area " + this.area();
    }
}

public class Rectangulo extends FiguraGeometrica {
    private double base;
    private double altura;

    public Rectangulo (double largo, double ancho) {
        super("Rectangulo");
        this.base = largo;
        this.altura = ancho;
    }

    public double area () {
        return this.base * this.altura;
    }
}
```

b) Defina la clase **TestAbstract** o desde donde utilice objetos **Rectangulo** para probar la utilidad de las clases abstractas.

```
1  public class TestAbstracto {
2      public static void main(String[] args) {
3          Rectangulo r1 = new Rectangulo(12.5, 23.7);
4          Rectangulo r2 = new Rectangulo(8.6, 33.1);
5
6          System.out.println("Area de r1 = " + r1.area());
7          System.out.println("Area de r2 = " + r2.toString());
8
9          if (r1.mayorQue(r2)) {
10             System.out.println("El rectángulo de mayor área es r1");
11         } else {
12             System.out.println("El rectángulo de mayor área es r2");
13         }
14     }
15 }
```

EJERCICIOS:

1.- Relación de composición:

a) Agregue el atributo tipoCliente (char), el mismo que puede ser: C (cliente),B (banca exclusiva), E (empresarial).

```
private char tipoCliente;
```

b) Implemente el código que sea necesario para asignar un tipo concreto a una Persona. Si no se especifica el tipo de cliente, por defecto se le asigna el tipo C.

```java
public void asignarTipoCliente(char tipo) {
    if (tipo == 'C' || tipo == 'B' || tipo == 'E') {
        this.tipoCliente = tipo;
    } else {
        System.out.println("Tipo de cliente no válido. Usando tipo por defecto 'C'.");
        this.tipoCliente = 'C';
    }
}
```

c) En el constructor de **Persona**, se debe de definir el numero de la cuenta del objeto persona previo a la creación del mismo. El número de la cuenta depende del tipo de cliente, en donde el primer carácter corresponde a este tipo (C,B,E) seguido de un número correlativo que inicia en 1000 para los tipos C, en 5000 para los tipos B y 8000 para los E. Por ejemplo el número de cuenta para una persona que es el primero de tipo C, será “C1000”.

```java
private void asignarNumeroCuenta() {
    int numeroBase;
    if (tipoCliente == 'C') {
        numeroBase = 1000;
    } else if (tipoCliente == 'B') {
        numeroBase = 5000;
    } else {
        numeroBase = 8000;
    }
    this.numeroCuenta = tipoCliente + Integer.toString(numeroBase);
}
```

d) Implemente los métodos **getter** y **setter** de ambas clase.

e) Implemente los métodos **retirar(cantidad)** y **depositar(cantidad)**, que permitiran depositar o retirar una determinada cantidad del saldo de una cuenta. Tenga en cuenta que lo mínimo que puede tener una cuenta es 50 soles. Por lo tanto, además realice las modificaciones correspondientes a los constructores a fin de cumplir con esta restricción.

```java
public void depositar(double cantidad) {
    if (cantidad > 0) {
        this.saldo += cantidad;
    } else {
        System.out.println("La cantidad a depositar debe ser mayor que cero.");
    }
}

public void retirar(double cantidad) {
    if (cantidad > 0) {
        if (this.saldo - cantidad >= 50.0) {
            this.saldo -= cantidad;
        } else {
            System.out.println("El saldo mínimo de la cuenta debe ser de 50 soles. No se puede retirar esta cantidad.");
        }
    } else {
        System.out.println("La cantidad a retirar debe ser mayor que cero.");
    }
}
```

f) Implemente el método **toString()** de ambas clases deben de permitir retornar la información de una persona de la siguiente manera:

```java
@Override
public String toString() {
    return "Cliente: " + clienteID + "\nTipo: " + tipoCliente + "\nNombres: " + nombre + " " + apellido +
      "\nNro Cuenta: " + numeroCuenta + "\nSaldo: " + String.format("%.2f", saldo);
}
```

```java
@Override
public String toString() {
    return "Número de Cuenta: " + numeroCuenta + "\nTitular:\n" + titular + "\nSaldo: " + String.format("%.2f", titular.getSaldo());
}
}
```

g) En la clase **TestComposicion** escriba las instrucciones para realizar las pruebas que evidencien lo realizado.

```java
public class TestComposicion {
    public static void main(String[] args) {
        Persona persona1 = new Persona(1, "Jair", "Ccama", 'C');
        Cuenta cuenta1 = new Cuenta("C1000", persona1);
        cuenta1.depositar(500);
        cuenta1.retirar(200);

        System.out.println("Información de la Persona:");
        System.out.println(persona1);

        System.out.println("\nInformación de la Cuenta:");
        System.out.println(cuenta1);

        persona1.asignarTipoCliente('A');
        Cuenta cuenta2 = new Cuenta("A5000", persona1);
        cuenta2.depositar(1000);
        cuenta2.retirar(300);

        System.out.println("\nInformación de la Nueva Cuenta:");
        System.out.println(cuenta2);
    }
}
```

2.- Relación de agregación:

a) Modifique el método **toString()** de ambas clases de modo que retornen la información de un automóvil (con o sin motor) y de un motor tal y como se observa a continuación:

```java
@Override
public String toString() {
    if (motor != null) {
        return "Automovil con motor\nPlaca: " + placa + "\nNúmero de puertas: " + numPuertas + "\nMarca: " + marca + "\nModelo: " + modelo + "\n" + motor;
    } else {
        return "Automovil sin motor\nPlaca: " + placa + "\nNúmero de puertas: " + numPuertas + "\nMarca: " + marca + "\nModelo: " + modelo;
    }
}
```

b) Realicelas pruebas enla clase **TestAgregación** que evidencien lo realizado.

```java
public class TestAgregacion {

    public static void main(String[] args) {
        // Crear un automóvil con motor
        Motor motor1 = new Motor(25, 100);
        Automovil automovilConMotor = new Automovil("A85AC", 4, "TOYOTA", "Camry", motor1);

        // Crear un automóvil sin motor
        Automovil automovilSinMotor = new Automovil("AU398", 3, "Toyota", "Corolla", null);

        // Mostrar información de ambos automóviles
        System.out.println(automovilConMotor);
        System.out.println();
        System.out.println(automovilSinMotor);
    }
}
```

3.- Relación de asociasión:

a) Agregar el método **darBajaCliente** de la clase **Banco**, que elimine a un determinado cliente en caso exista, y retorne al cliente que se esta eliminando. El espacio que queda vacío debe ser ocupado por el cliente que está la ultima posición ocupada.

```java
public Persona darBajaCliente(Persona persona) {
    int indice = -1;
    for (int i = 0; i < cantidadClientes; i++) {
        if (clientes[i].equals(persona)) {
            indice = i;
            break;
        }
    }

    if (indice != -1) {
        Persona clienteEliminado = clientes[indice];
        clientes[indice] = clientes[cantidadClientes - 1];
        clientes[cantidadClientes - 1] = null;
        cantidadClientes--;
        return clienteEliminado;
    } else {
        return null;
    }
}
```

b) Agregar el método **clienteTipo** de la clase **Banco**, que liste los datos de los clientes y su cuenta de un tipo determinado: C (cliente), B (banca exclusiva), E (empresarial).

```java
    public void clienteTipo(char tipo) {
        System.out.println("Clientes de tipo " + tipo + " en el banco " + nombre + ":");
        for (int i = 0; i < cantidadClientes; i++) {
            Persona cliente = clientes[i];
            char tipoCliente = obtenerTipoCliente(cliente);
            if (tipoCliente == tipo) {
                System.out.println(cliente.toString());
            }
        }
    }
```

c) En la clase **TestAsociacion** pruebe los métodos que se han adicionado para eliminar clientes y mostrar los clientes por los diferentes tipos.

```java
public class TestAsociacion {

    public static void main(String[] args) {
        Banco bcp = new Banco("BCP");
        Banco bbva = new Banco("BBVA", 30);
        Scanner scanner = new Scanner(System.in);

        bcp.agregarCliente(new Persona("Andrea Perez"));
        bcp.agregarCliente(new Persona("Maria Bolaños"));
        bbva.agregarCliente(new Persona("Carlos Lopez"));

        bcp.mostrarClientes();

        bbva.mostrarClientes();

        System.out.println("Ingrese el nombre del cliente a buscar en el BCP:");
        String nombreCliente = scanner.nextLine();
        boolean encontrado = bcp.buscarCliente(new Persona(nombreCliente));
        if (encontrado) {
            System.out.println("El cliente " + nombreCliente + " se encuentra en el BCP.");
        } else {
            System.out.println("El cliente " + nombreCliente + " no se encuentra en el BCP.");
        }

        System.out.println("Ingrese el nombre del cliente a dar de baja en el BCP:");
        String clienteABaja = scanner.nextLine();
        Persona clienteEliminado = bcp.darBajaCliente(new Persona(clienteABaja));
        if (clienteEliminado != null) {
            System.out.println("Se ha dado de baja al cliente: " + clienteEliminado.getNombre());
        } else {
            System.out.println("El cliente " + clienteABaja + " no se encontró en el BCP.");
        }

        System.out.println("Ingrese el tipo de cliente a listar (C/B/E):");
        char tipoCliente = scanner.nextLine().charAt(0);
        bcp.clienteTipo(tipoCliente);

        scanner.close();
    }
}
```

4.- Relaciones de herencia:

Sean las clases vistas en la actividad 4. Imagene que, se debe agregar un nuevo método que permita calcular la nómica de un empleado. Todos los empleados asallariados de la firma tienen un atributo que representa su saliro base. Los empleados de producción reciben sobre esta cantidad un incemento del 15%, mientras que los empleados de distribución reciben un 10% de incremento.

a) Modifique las clases de modo que se cumple el requerimiento antes descrito.
b) Agregue el método **toString()** a las clases de modo que la cadena que retornen contenga la siguiente información según sea el caso y de la forma como se muestra:
c) En la clase **TestHerencia** escriba las instrucciones que permitan evidencias las modificaciones realizadas.

```java
public class TestHerencia {

    public static void main(String[] args) {
        Asalariado empleado1 = new Asalariado("Manuel Cortina", 12345678, 28, 2000.0);
        EmpleadoProduccion empleado2 = new EmpleadoProduccion("noche", 55333222, 30, 2500.0);
        EmpleadoDistribucion empleado3 = new EmpleadoDistribucion("Granada", 55333666, 35, 2200.0);

        double nominaEmpleado1 = empleado1.calcularNomina();
        double nominaEmpleado2 = empleado2.calcularNomina();
        double nominaEmpleado3 = empleado3.calcularNomina();

        System.out.println("Nómina del empleado 1: " + nominaEmpleado1);
        System.out.println("Nómina del empleado 2: " + nominaEmpleado2);
        System.out.println("Nómina del empleado 3: " + nominaEmpleado3);

        System.out.println(empleado1.toString());
        System.out.println(empleado2.toString());
        System.out.println(empleado3.toString());
    }
}
```