UVM | Universidad del Valle de México
LAUREATE INTERNATIONAL UNIVERSITIES®

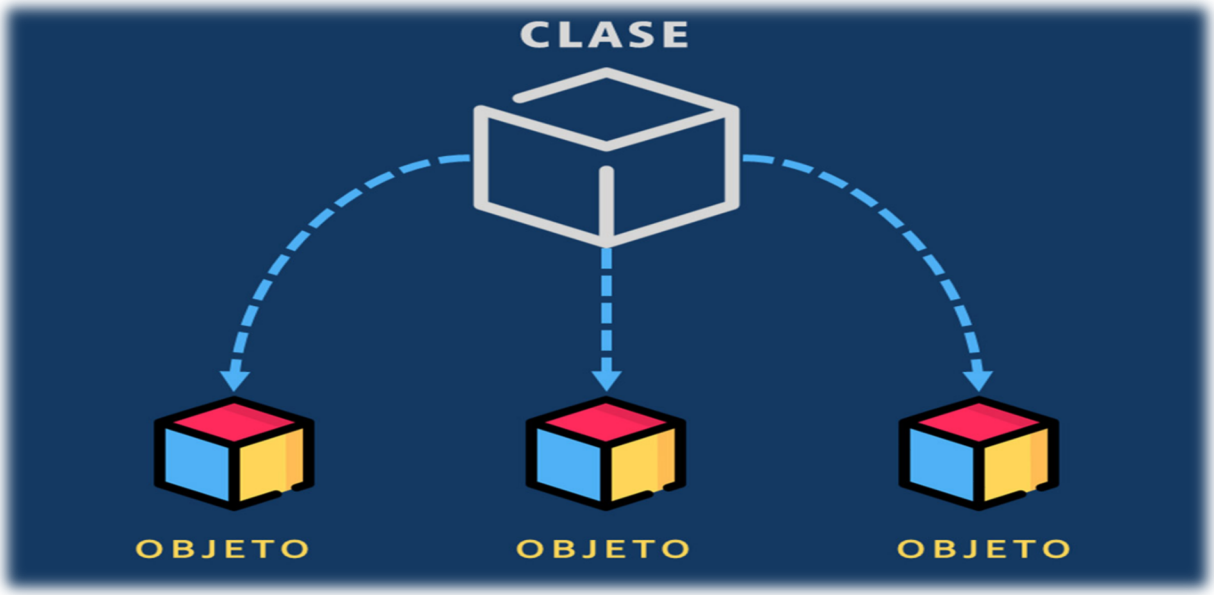
Actividad 8
Ejercicio

Programación orientada a objetos

Docente. Ing. Saúl Santiago Rivera

José Emiliano Jauregui Guzmán

Por siempre responsable de lo que se ha cultivado



ACTIVIDAD VIII:

EJERCICIO

Con base en el material consultado en la unidad resuelve el siguiente ejercicio que se plantea a continuación acerca de los siguientes temas:

- Lenguaje C++
- Características orientadas a la POO (Herencia, abstracción, encapsulación)
- Métodos

Para realizar esta actividad se utilizará el entorno de desarrollo de Visual Studio para modelar la clase Persona y sus clases derivadas Cliente y Empleado.

Considera el siguiente requerimiento:
Un establecimiento necesita hacer un registro de la información básica de sus clientes y empleados.

Ejercicio. Registro de clientes de un banco

Sigue los pasos que se indican a continuación para realizar el ejercicio propuesto:

a) Define una clase base “Persona” con los atributos generales de una persona. Identifica por lo menos 3 atributos para la clase.

Clase Padre:
Persona

Atributos:
•Nombre
•Edad
•Documento de identidad. (DNI, RFC, Pasaporte, etc.)

b) Paso 2. Establece 2 clases derivadas que hereden de tu clase Persona. Estas clases son Cliente y Empleado. Identifica por lo menos 3 atributos específicos para cada clase.

Clase Cliente:

Atributos:
•Número de cuenta
•Tipo de cuenta
•Saldo de su cuenta

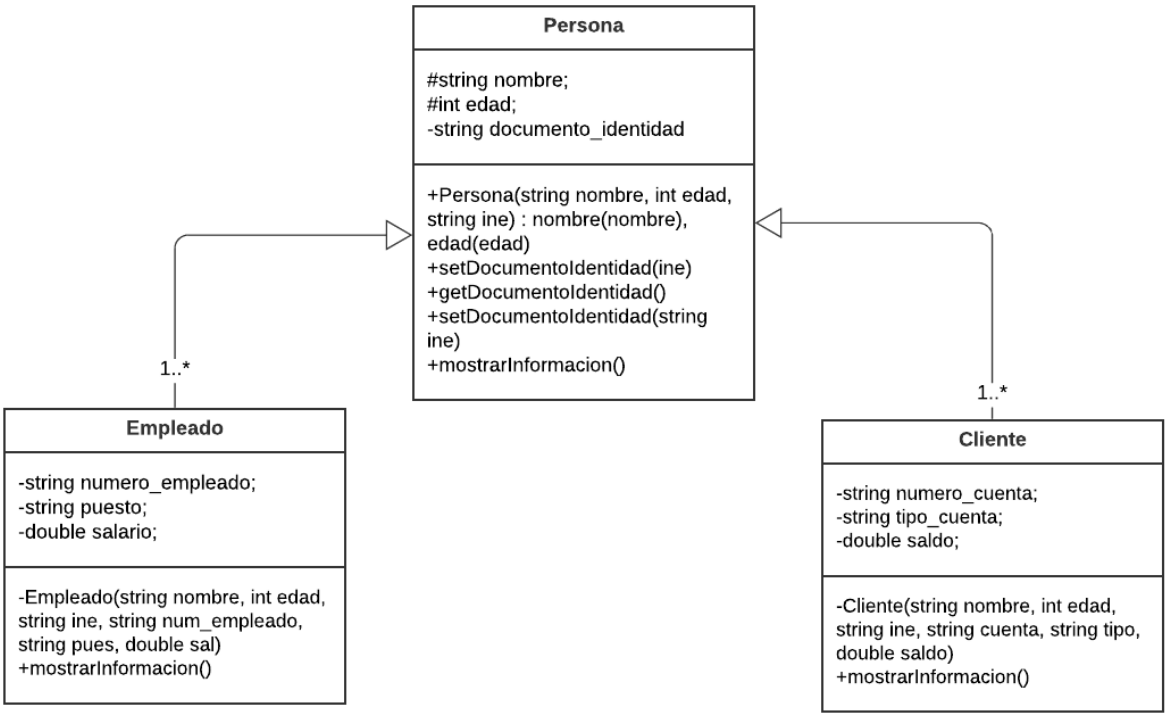
Clase Empleado:

Atributos:
•Número de empleado
•Puesto
•Salario

c) Para cada una de las clases hijas define un método registro, que reciba como parámetros los atributos de la clase y los asigne a los miembros privados.

1. Método registrar a un cliente.
2. Método registrar a un empleado.

d) Realiza el modelado de tus clases (Ver video sobre diagrama de clase UML: https://www.youtube.com/watch?v=Z0yLerU0g-Q)



e) Realiza un programa en Visual C++ donde implementes las clases utilizando **herencias**.

Para este punte estaré utilizando la herencia en el nombre y la edad para que las clases hijas puedan tener estas funciones sin necesidad de agregárselas individualmente con el fin de igual manera de hacer el código mas eficiente y preciso para la elaboración de esta actividad.

f) Utiliza **encapsulamiento** en alguna de las propiedades. (Ver video sobre Encapsulación: Programación en C++ 16 - Encapsulación y modificadores de acceso: https://www.youtube.com/watch?v=-9cHzKfcXoo)

Únicamente utilizare el encapsulamiento en el apartado de documento de identidad para hacer correctamente este punto.

g) En la función principal de tu programa *main* () solicita la captura de un cliente y un empleado y después imprime en pantalla los datos registrados.

```
#include <iostream>
#include <string>
using namespace std;

// Clase Pabre
class Persona {
protected:
    string nombre;
    int edad;

private:
    string documento_identidad; // Atributo encapsulado

public:
    // Constructor
    Persona(string nombre, int edad, string ine) : nombre(nombre), edad(edad) {
        setDocumentoIdentidad(ine);
    }

    // Métodos getter y setter para el atributo identidad
    string getDocumentoIdentidad() const {
        return documento_identidad;
    }

    void setDocumentoIdentidad(string ine) {
        if (ine.length() == 13) { // Longitud válida para el INE
            documento_identidad = ine;
        } else {
```

```
            cout << "INE inválido. Debe tener 13 caracteres." << endl;
        }
    }

    // Mostrar la información de la persona
    void mostrarInformacion() const {
        cout << "Nombre: " << nombre << endl;
        cout << "Edad: " << edad << endl;
        cout << "Documento de Identidad (INE): " << getDocumentoIdentidad() << endl; // Usamos
getter para acceder al encapsulamiento
    }
};

// Clase hija Cliente
class Cliente : public Persona {
private:
    string numero_cuenta;
    string tipo_cuenta;
    double saldo;

public:
    // Constructor clase Cliente
    Cliente(string nombre, int edad, string ine, string cuenta, string tipo, double saldo)
        : Persona(nombre, edad, ine), numero_cuenta(cuenta), tipo_cuenta(tipo), saldo(saldo) {}

    // Método para mostrar la información del cliente
    void mostrarInformacion() const {
        Persona::mostrarInformacion(); // Llamar a clase padre
        cout << "Número de cuenta: " << numero_cuenta << endl;
        cout << "Tipo de cuenta: " << tipo_cuenta << endl;
        cout << "Saldo en la cuenta: $" << saldo << endl;
    }
};

// Clase hija Empleado
```

```
class Empleado : public Persona {
private:
    string numero_empleado;
    string puesto;
    double salario;

public:
    // Constructor de la clase Empleado
    Empleado(string nombre, int edad, string ine, string num_empleado, string pues, double sal)
        : Persona(nombre, edad, ine), numero_empleado(num_empleado), puesto(pues),
salario(sal) {}

    // Método para mostrar la información del empleado
    void mostrarInformacion() const {
        Persona::mostrarInformacion(); // Llamar a clase padre
        cout << "Número de empleado: " << numero_empleado << endl;
        cout << "Puesto: " << puesto << endl;
        cout << "Salario: $" << salario << endl;
    }
};

// Función principal
int main() {

    Cliente cliente1("Jose Emiliano", 23, "1234567890123", "00123456789", "Inversion",
199500.99);

    Empleado empleado1("Valeria Rico", 22, "1234567890123", "EMP123", "Asesor", 35000.55);

    cout << "\nInformación del Cliente:" << endl;
    cliente1.mostrarInformacion();

    cout << "\nInformación del Empleado:" << endl;
    empleado1.mostrarInformacion();
    return 0;
}
```

main.cpp

```cpp
#include <iostream>
#include <string>
using namespace std;

// Clase Padre
class Persona {
protected:
    string nombre;
    int edad;

private:
    string documento_identidad; // Atributo encapsulado

public:
    // Constructor
    Persona(string nombre, int edad, string ine) : nombre(nombre), edad(edad) {
        setDocumentoIdentidad(ine);
    }

    // Métodos getter y setter para el atributo identidad
    string getDocumentoIdentidad() const {
        return documento_identidad;
    }

    void setDocumentoIdentidad(string ine) {
        if (ine.length() == 13) { // Longitud válida para el INE
            documento_identidad = ine;
        } else {
            cout << "INE inválido. Debe tener 13 caracteres." << endl;
        }
    }

    // Mostrar la información de la persona
    void mostrarInformacion() const {
        cout << "Nombre: " << nombre << endl;
        cout << "Edad: " << edad << endl;
        cout << "Documento de Identidad (INE): " << getDocumentoIdentidad() << en
    }
};

// Clase hija Cliente
class Cliente : public Persona {
private:
    string numero_cuenta;
    string tipo_cuenta;
    double saldo;

public:
    // Constructor clase Cliente
    Cliente(string nombre, int edad, string ine, string cuenta, string tipo, doub
        : Persona(nombre, edad, ine), numero_cuenta(cuenta), tipo_cuenta(tipo), s

    // Método para mostrar la información del cliente
    void mostrarInformacion() const {
        Persona::mostrarInformacion(); // Llamar a clase padre
        cout << "Número de cuenta: " << numero_cuenta << endl;
        cout << "Tipo de cuenta: " << tipo_cuenta << endl;
        cout << "Saldo en la cuenta: $" << saldo << endl;
    }
};

// Clase hija Empleado
class Empleado : public Persona {
private:
    string numero_empleado;
    string puesto;
    double salario;

public:
    // Constructor de la clase Empleado
    Empleado(string nombre, int edad, string ine, string num_empleado, string pue
        : Persona(nombre, edad, ine), numero_empleado(num_empleado), puesto(pues

    // Método para mostrar la información del empleado
    void mostrarInformacion() const {
        Persona::mostrarInformacion(); // Llamar a clase padre
        cout << "Número de empleado: " << numero_empleado << endl;
        cout << "Puesto: " << puesto << endl;
        cout << "Salario: $" << salario << endl;
    }
};

// Función principal
int main() {

    Cliente cliente1("Jose Emiliano", 23, "1234567890123", "00123456789", "Inver

    Empleado empleado1("Valeria Rico", 22, "1234567890123", "EMP123", "Asesor",

    cout << "\nInformación del Cliente:" << endl;
    cliente1.mostrarInformacion();

    cout << "\nInformación del Empleado:" << endl;
    empleado1.mostrarInformacion();

    return 0;
}
```

```
Información del Cliente:
Nombre: Jose Emiliano
Edad: 23
Documento de Identidad (INE): 1234567890123
Número de cuenta: 00123456789
Tipo de cuenta: Inversion
Saldo en la cuenta: $199501

Información del Empleado:
Nombre: Valeria Rico
Edad: 22
Documento de Identidad (INE): 1234567890123
Número de empleado: EMP123
Puesto: Asesor
Salario: $35000.6


...Program finished with exit code 0
Press ENTER to exit console.
```

• Redacta una conclusión en la que expliques la utilidad de la herencia y la jerarquía de clases y cómo la aplicas en el diseño de tu programa.

La herencia y la jerarquía de clases son muy importantes mas que nada en esta materia ya que me permitió a lo largo de la actividad estructurar desde el inicio el código con el que fui trabajando haciendo mi desempeño mucho mas eficiente y no tener que volver a retomar ciertos puntos del código como lo fue en la clase padre persona que me ayudo a tener las bases de nombre y edad como atributos que fueron heredadas por las clases hijas, ya que me permite tender como base esos atributos y yo después agregar mas por separado lo que me ayudo a reducir el tiempo de trabajo. Me demuestra este tipo de ejercicio que si quiero modificar un atributo en especifico sabré que pudo hacerlo desde la clase padre porque asi se vera reflejado en las clases hijas; en este mismo código al derivar que un empleado y un cliente son personas simplifica la comprensión del tema. Por ello la importancia de la herencia hace mas fácil y optimo la estructura de mi programa y me ayuda a ser mas organizado.

Referencias

Cervantes, J., Gómez, M. (2016). Introducción a la programación orientada a objetos Haga clic para ver más opciones Recuperado el 6 de diciembre del 2024, de https://www.cua.uam.mx/pdfs/revistas_electronicas/libros-electronicos/2016/2intro-poo/programacion_web.pdf

Oviedo, E. (2015). Lógica de programación orientada a objetos Haga clic para ver más opciones. Recuperado el 6 de diciembre del 2024, de https://www.studocu.com/co/document/pontificia-universidad-javeriana/taller-de-programacion/logica-de-programacion-orientada-a-objetos-cap-7-pg-219-287/17217649?origin=home-recent-3

Fredy Geek (Productor). (13 de Febrero de 2019). ¿Qué es la Herencia? - Programación Orientada a Objetos. Recuperado el 6 de diciembre del 2024, de https://www.youtube.com/watch?v=9NynVRpZzv4

Fredy Geek (Productor). (13 de Febrero de 2019). ¿Qué es la Herencia? - Programación Orientada a Objetos. Recuperado el 6 de diciembre del 2024, de https://www.youtube.com/watch?v=9NynVRpZzv4

Fredy Geek (Productor). (04 de Abril de 2018). ¿Qué es el Encapsulamiento? – Programación Orientada a Objetos. Recuperado el 6 de diciembre del 2024, de https://www.youtube.com/watch?v=gR0EssHrl24&t=84s

Fredy Geek (Productor). (30 de Marzo de 2018). ¿Qué es la Abstración? - Programación Orientada a Objetos. Recuperado el 6 de diciembre del 2024, de https://www.youtube.com/watch?v=0B001Cx_YwQ

YouTube. (n.d.-h). Youtu.Be. Recuperado el 6 de diciembre del 2024, de https://youtu.be/-9cHzKfcXoo?si=yPZHsxVz5YLYCQYa