Instituto Politécnico Nacional

Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas

“UPIITA”

Ingeniería Mecatrónica

Asignatura: Programación Avanzada

“Tema 3: Herencia”

Profesora: Maza Casas Lamberto

Grupo: 2MV3

Alumnos:

Alonso Bernal Alejandro

López Piedracruz Marcos Antonio

Luviano Murakawa Tsuioshi Alberto

**Introducción**

La herencia es después de la agregación o composición, el mecanismo más utilizado para alcanzar algunos de los objetivos más preciados en el desarrollo de software como lo son la reutilización y la extensibilidad. A través de ella los diseñadores pueden crear nuevas clases partiendo de una clase o de una jerarquía de clases preexistente evitando con ello el rediseño, la modificación y verificación de la parte ya implementada. La herencia facilita la creación de objetos a partir de otros ya existentes e implica que una subclase obtiene todo el comportamiento (métodos) y eventualmente los atributos (variables) de su superclase.

Es la relación entre una clase general y otra clase más específica. Por ejemplo: Si declaramos una clase párrafo derivada de una clase texto, todos los métodos y variables asociadas con la clase texto, son automáticamente heredados por la subclase párrafo.

La herencia es uno de los mecanismos de los lenguajes de [programación orientada a objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos) basados en clases, por medio del cual una [clase](https://es.wikipedia.org/wiki/Clase_(programaci%C3%B3n_orientada_a_objetos)) se deriva de otra de manera que extiende su funcionalidad. La clase de la que se hereda se suele denominar *clase base*, *clase padre*, *superclase*, *clase ancestro* (el vocabulario que se utiliza suele depender en gran medida del lenguaje de programación).

En los lenguajes que cuentan con un [sistema de tipos](https://es.wikipedia.org/wiki/Sistema_de_tipos) fuerte y estrictamente restrictivo con el tipo de datos de las variables, la herencia suele ser un requisito fundamental para poder emplear el [Polimorfismo](https://es.wikipedia.org/wiki/Polimorfismo_(programaci%C3%B3n_orientada_a_objetos)), al igual que un mecanismo que permita decidir en tiempo de ejecución qué método debe invocarse en respuesta a la recepción de un mensaje, conocido como *enlace tardío* (*late binding*) o *enlace dinámico* (*dynamic binding*).

Existen 3 tipos de herencia, que son las secciones privadas, públicas y protegidas.

**Las herencias públicas** significan que una clase derivada tiene acceso a los elementos públicos y protegidos de su clase base. La diferencia que llega a tener con los protegidos es que los elementos públicos se heredan como elementos públicos; los elementos protegidos permanecen protegidos.

**Las herencias privadas** significan que un usuario de la clase derivada no tiene acceso a ninguno de sus elementos de su clase base. Con la herencia privada, los miembros públicos y protegidos de la base se vuelven miembros privados de la clase derivada, los usuarios de clases privadas no tienen acceso a las facilidades proporcionadas por la clase base.

**Las herencias protegidas** los elementos públicos y protegidos de la clase base se convierten en miembros protegidos de la clase derivada y los miembros privados de la clase se vuelven inaccesibles, se apropia de cuando las facilidades o aptitudes de la clase base son útiles en la implementación de una clase derivada, pero no son parte de la interfaz que el usuario de la clase utiliza.

**DESARROLLO**

*Empleado.cpp - Resuelve los métodos de la clase Empleado*

#include <iostream>

using std::cout;

using std::endl;

#include <cstring>

using std::strlen;

using std::strcpy;

#include "Empleado.h"

int Empleado::cuenta=0;

int Empleado::getCuenta(){

return cuenta;

}

Empleado::Empleado(const char \*const nombre,const char \*const apellido){

primerNombre=new char[strlen(nombre)+1];

strcpy(primerNombre,nombre);

apellidoPaterno=new char[strlen(apellido)+1];

strcpy(apellidoPaterno,apellido);

cuenta++;

cout<<"Se llam\\'o al constructor de empleado para "<<primerNombre<<" "

<<apellidoPaterno<<"."<<endl;

}

*El destructor designa a la memoria asignada en forma dinámica*

Empleado::~Empleado(){

cout<<"Se llamó a ~Empleado() para "<<primerNombre

<<' '<<apellidoPaterno<<endl;

delete [] primerNombre;

delete [] apellidoPaterno;

cuenta--;

}

*TestEmpleado.cpp - Usa la clase Empleado*

#include <iostream>

using std::cout;

using std::endl;

#include "Empleado.h"

int main()

{

cout<<"El n\\'umero de empleados antes de instanciar cualquier objeto es "

<<Empleado::getCuenta()<<endl;

Empleado \*e1Ptr=new Empleado("Susan","Baker");

Empleado \*e2Ptr=new Empleado("Robert","Jones");

cout<<"El n\\'umero de empleados despu\\'es de instanciar \

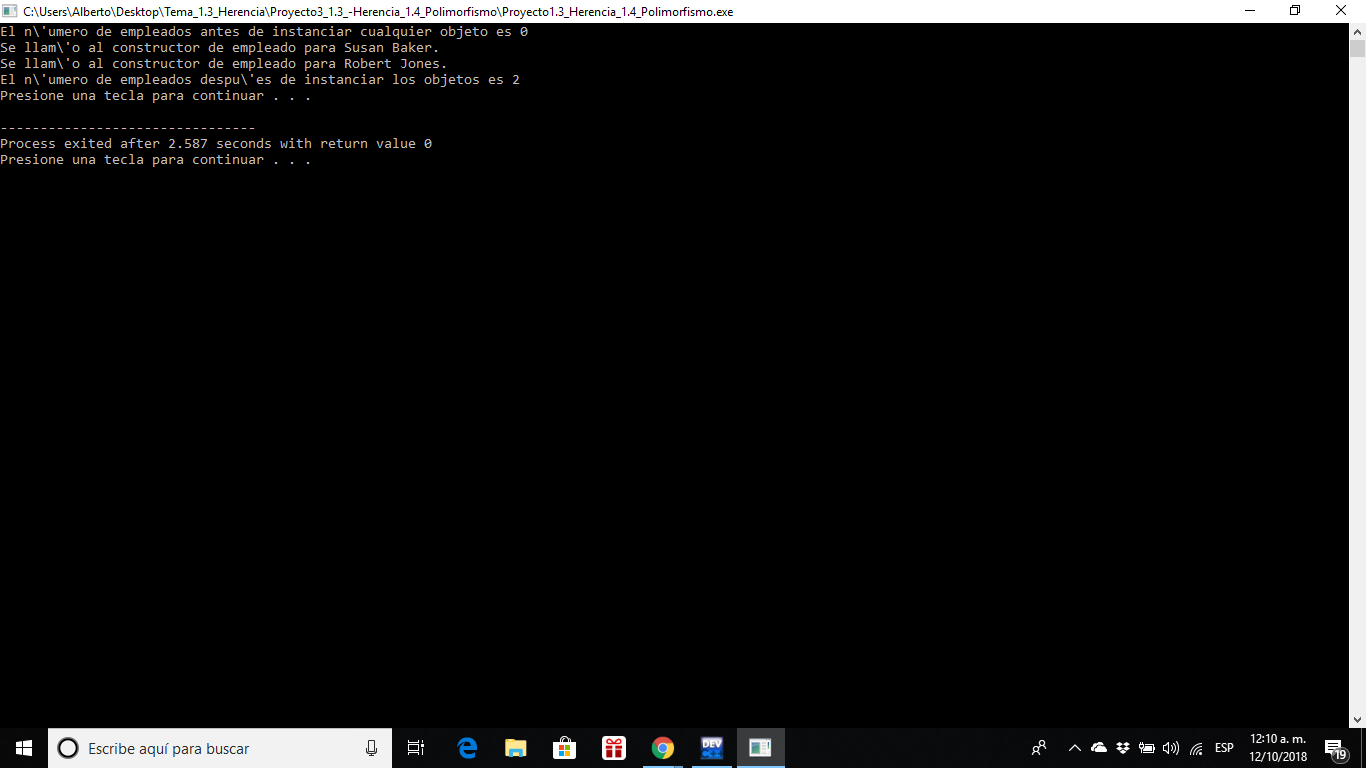
los objetos es "<<Empleado::getCuenta()<<endl;

system("pause");

return 0;

}

**EVIDENCIAS**



**CONCLUSIÓN**

Concluyendo la herencia es una parte importante para la programación ya que es posible crear jerarquías de clases relacionadas y reduce la cantidad de código redundante en componentes de la clase.

También tiene propiedades que permite definir nuevas clases usando como base clases ya existentes, las cuales pueden heredar los atributos y comportamientos que son específicos de ella.