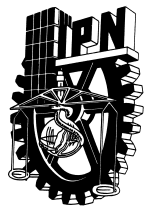
** INSTITUTO POLITECNICO NACIONAL**

**UNIDAD PROFESIONAL INTERDISCIPLINARIA**

**EN INGENIERIA Y TECNOLOGIAS**

**AVANZADAS**

Fecha: 05/12/18

Unidad de Aprendizaje: Programación Avanzada

Practica 5

Alumnos

Salgado Ríos Oscar David

García Morales Pablo

Ruíz Méndez Néstor Alberto:

Grupo: 2MV3

Profesor: Maza Casas Lamberto

**Introducción**

En programación orientada a objetos se denomina polimorfismo a la capacidad que tienen los objetos de una clase de responder al mismo mensaje o evento en función de los parámetros utilizados durante su invocación. Un objeto polimórfico es una entidad que puede contener valores de diferentes tipos durante la ejecución del programa.

En algunos lenguajes, el término polimorfismo es también conocido como ‘Sobrecarga de parámetros’ ya que las características de los objetos permiten aceptar distintos parámetros para un mismo método generalmente con comportamientos distintos e independientes para cada una de ellas.

En general, hay tres tipos de polimorfismos: el polimorfismo de sobre carga, el polimorfismo paramétrico y el polimorfismo de inclusión.

**Polimorfismo de sobrecarga:**

El polimorfismo de sobrecarga ocurre cuando las funciones del mismo nombre existen, con función similar, en clases que son completamente independientes unas de otras. El polimorfismo de sobrecarga nos permite definir operadores cuyos comportamientos varían de acuerdo a los parámetros que se les aplican.

**Polimorfismo paramétrico:**

El polimorfismo paramétrico es la capacidad para definir varias funciones utilizando el mismo nombre, pero usando parámetros diferentes. El polimorfismo paramétrico selecciona automáticamente el método correcto a aplicar en función del tipo de datos pasados en el parámetro.

Por lo tanto, podemos definir varios métodos homónimos de addition() efectuando una suma de valores. El método int adddition(int,int) devolvería la suma de dos números enteros. Por su parte, float addition(float,float) devolvería la suma de dos flotantes. En cuanto a char addition(char,char) daría por resultado la suma de dos caracteres definidos por el autor.

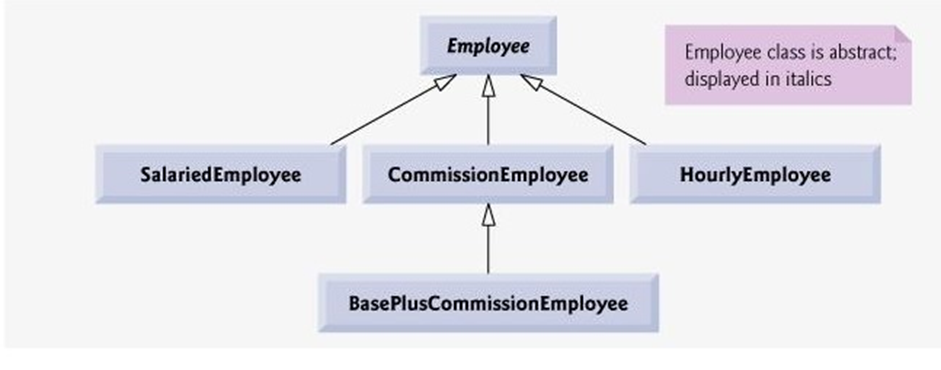
Una signature es el nombre y tipo (estático) que se da a los argumentos de una función. Por esto, una firma de método determina qué elemento se va a llamar.

**Polimorfismo de subtipado:**

La habilidad para redefinir un método en clases que se hereda de una clase base se llama especialización. Por lo tanto, se puede llamar un método de objeto sin tener que conocer su tipo intrínseco: esto es polimorfismo de subtipado. Permite no tomar en cuenta detalles de las clases especializados de una familia de objetos, enmascarándolos con una interfaz común (siendo esta la clase básica).

**Desarrollo:**

Se diseñó un proyecto llamado Empleado, dentro de este proyecto, a partir de una clase abstracta llamada “Empleado” con métodos para imprimir sus datos (nombre, apellido, NSS, etc.) y para definir su salario, se definen otro tipo de clases con los mismos nombres para sus métodos específicos (aplicando el concepto de polimorfismo) definiendo de esta manera distintas formas para calcular el salario de un empleado.



EMPLEADO ASALARIADO

EmpleadoAsalariado::EmpleadoAsalariado(const string& nombre,

const string& apellido,

const string& nss, double salario)

: Empleado(nombre,apellido,nss)

{

setSalarioSemanal(salario);

}

EmpleadoAsalariado::~EmpleadoAsalariado()

{ }

void EmpleadoAsalariado::setSalarioSemanal(double salario) {

salarioSemanal = (salario < 0.0) ? 0.0 : salario;

}

double EmpleadoAsalariado::getSalarioSemanal() const {

return salarioSemanal;

}

//sobrescribe la funci\'on virtual

double EmpleadoAsalariado::ingresos() const {

return getSalarioSemanal();

}

void EmpleadoAsalariado::imprimir() const {

cout << "Empleado asalariado: ";

Empleado::imprimir();//reutiliza la funci\'on imprimir

//de la clase base abstracta

cout << "\nSalario semanal: " << getSalarioSemanal();

}

EMPLEADO POR COMISION

|  |
| --- |
| #include "stdafx.h" |
|  |

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| using std::cout; |
|  |

|  |
| --- |
| #include "EmpleadoPorComision.h" |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| //constructor |
|  |

|  |
| --- |
| EmpleadoPorComision::EmpleadoPorComision(const string& nombre, |
|  |

|  |
| --- |
| const string& apellido, const string&nss, |
|  |

|  |
| --- |
| double ventas, double tarifa) |
|  |

|  |
| --- |
| :Empleado(nombre,apellido,nss) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| //primerNombre = nombre; |
|  |

|  |
| --- |
| //apellidoPaterno = apellido; |
|  |

|  |
| --- |
| //numeroSeguroSocial = nss; |
|  |

|  |
| --- |
| setVentasBrutas(ventas); |
|  |

|  |
| --- |
| setTarifaComision(tarifa); |
|  |

|  |
| --- |
| }//end constructor EmpleadoPorComision |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| EmpleadoPorComision::~EmpleadoPorComision() |
|  |

|  |
| --- |
| { } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| //void EmpleadoPorComision::setPrimerNombre(const string& nombre) { |
|  |

|  |
| --- |
| // primerNombre = nombre; |
|  |

|  |
| --- |
| //} |
|  |

|  |
| --- |
| // |
|  |

|  |
| --- |
| //string EmpleadoPorComision::getPrimerNombre() const { |
|  |

|  |
| --- |
| // return primerNombre; |
|  |

|  |
| --- |
| //} |
|  |

|  |
| --- |
| // |
|  |

|  |
| --- |
| //void EmpleadoPorComision::setApellidoPaterno(const string& apellido) { |
|  |

|  |
| --- |
| // apellidoPaterno = apellido; |
|  |

|  |
| --- |
| //} |
|  |

|  |
| --- |
| // |
|  |

|  |
| --- |
| //string EmpleadoPorComision::getApellidoPaterno() const { |
|  |

|  |
| --- |
| // return apellidoPaterno; |
|  |

|  |
| --- |
| //} |
|  |

|  |
| --- |
| // |
|  |

|  |
| --- |
| //void EmpleadoPorComision::setNumeroSeguroSocial(const string& nss) { |
|  |

|  |
| --- |
| // numeroSeguroSocial = nss; |
|  |

|  |
| --- |
| //} |
|  |

|  |
| --- |
| // |
|  |

|  |
| --- |
| //string EmpleadoPorComision::getNumeroSeguroSocial() const { |
|  |

|  |
| --- |
| // return numeroSeguroSocial; |
|  |

|  |
| --- |
| //} |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void EmpleadoPorComision::setVentasBrutas(double ventas) { |
|  |

|  |
| --- |
| ventasBrutas = (ventas<0.0) ? 0.0 : ventas; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| double EmpleadoPorComision::getVentasBrutas() const { |
|  |

|  |
| --- |
| return ventasBrutas; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void EmpleadoPorComision::setTarifaComision(double tarifa) { |
|  |

|  |
| --- |
| tarifaComision = (tarifa > 0.0 && tarifa < 1.0) ? tarifa : 0.0; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| double EmpleadoPorComision::getTarifaComision() const { |
|  |

|  |
| --- |
| return tarifaComision; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| //calcula los ingresos |
|  |

|  |
| --- |
| double EmpleadoPorComision::ingresos() const { |
|  |

|  |
| --- |
| //return tarifaComision \* ventasBrutas; |
|  |

|  |
| --- |
| return getTarifaComision()\*getVentasBrutas(); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| //imprime el objeto EmpleadoPorComision |
|  |

|  |
| --- |
| void EmpleadoPorComision::imprimir() const { |
|  |

|  |
| --- |
| /\*cout << "Empleado por comisi¢n: " << primerNombre << ' ' |
|  |

|  |
| --- |
| << apellidoPaterno << "\nN£mero de seguro social: " |
|  |

|  |
| --- |
| << numeroSeguroSocial << "\nVentas brutas: " |
|  |

|  |
| --- |
| << ventasBrutas << "\nTarifa de comisi¢n: " |
|  |

|  |
| --- |
| << tarifaComision;\*/ |
|  |

|  |
| --- |
| cout << "Empleado por comisi¢n: "; |
|  |

|  |
| --- |
| Empleado::imprimir(); //reutilizaci\'on de c\'odigo |
|  |

|  |
| --- |
| cout << "\nVentas brutas: " << getVentasBrutas() |
|  |

|  |
| --- |
| << "; tarifa de comisi¢n: " << getTarifaComision(); |
|  |
|  |

}

**Conclusión**

En este trabajo utilizando los polimorfismos concluimos que al ocupar una variable para diferentes objetivos nos ayuda definir una variable multifuncional y que no ocupe los detalles en la herencia de la clase.