TRABAJO PRACTICO N: 9.

Tema: Programación orientada a objetos

Objetivos:

* Comprender los conceptos básicos del paradigma orientado a objetos.
* Diferenciar el paradigma orientado a objetos del paradigma procedimental.
* Comprender la definición de COMPONENTE.
* Discriminar los atributos de un objeto de sus procedimientos.
* Entender la abstracción de datos
* Comprender y dominar la representación de clases
* Aprender a crear objetos a partir de clases.
* Resolver problemas mediante el uso del paradigma orientado a objetos
* Comprender la funcionalidad de los métodos: constructor y destructor
* Aprender a usar funciones amigas
* Comprender la funcionalidad de las clases amigas
* Comprender los conceptos básicos de la herencia simple y múltiple
* Aprender a utilizar la sintaxis de C++ para herencia simple y múltiple.
* Comprender los conceptos de polimorfismo.
* Reconocer clases abstractas y sus propósitos
* Entender la sobrecarga de operadores
* Aprender a usar y reconocer la sintaxis de la sobrecarga de operadores

PROGRAMACIÓN ORIENTADA A OBJETOS

ABSTRACCIÓN DE DATOS

La abstracción de datos (TDA) se entiende como la extracción de las propiedades esenciales de un concepto. Con la abstracción de datos se puede aislaran la representación o estructura de un objeto de los detalles de la implementación del mismo, solo interesa que es el objeto y como se comporta.

DISEÑO DE CLASES Y OBJETOS

Hasta aquí en programación trabajamos el paradigma procedimental, ahora trabajaremos otro paradigma: Paradigma Orientado a Objetos (POO).

Los programas POO, se organizan a través de objetos y clases, los cuales contienen datos (atributos) y los métodos que actúan sobre los datos, determinando el comportamiento del objeto.

Una clase es un tipo de dato que contienen código (métodos) y datos (atributos). Una clase permite encapsular todo el código y los datos necesarios para gestionar un tipo de dato específico de un elemento de programa, como una figura gráfica en pantalla de un programa, un dispositivo conectado a una PC, o por ejemplo una ventana en windows es un objeto de tipo específico.

La programación orientada a objetos (POO) es en realidad una forma de programación muy parecida a como nosotros hacemos cosas. POO se caracteriza por las siguientes propiedades: objetos, Clases, Encapsulamiento, Herencia y polimorfismo.

Como idea preliminar para formar un esquema mental que nos ayude a pensar en términos del paradigma que estamos presentando, diremos que un programa orientado a objetos está formado por una colección de objetos interactuando conjuntamente para representar y solucionar algún problema.

En la figura 1 se muestra un programa orientado a objetos, en el esquema se observa un conjunto de objetos relacionados entre sí, estos permiten hacer entrada de datos, cálculos para resolver el problema, y generar las salidas de datos transformados en información.

Objeto 1

Figura 1

Objeto 2

Objeto 3

Objeto 6

Objeto 4

Objeto 7

Objeto 5

Objeto 8

Entonces ¿qué son los objetos?

Los objetos son entes, entidades, sujetos o cosas en el dominio del problema de nuestra realidad.

¿Cómo están formados los objetos?

Los objetos están compuestos por partes:

1. Atributos que describen la esencia del objeto. Son los datos
2. Métodos que representan la dinámica del objeto es decir las acciones. Estos métodos se implementan bajo la forma de funciones, procedimientos, etc.

En un diagrama de clases podemos representar un objeto con un rectángulo de bordes redondeados, como se muestra en la figura 2.

Cómo identificamos objetos?

Cuando se estudia el problema a resolver, hay que identificar los objetos involucrados y sus relaciones.

Objeto 1

Figura 2

dato1

dato2

dato3

……..

datoN

método1( )

método2 ( )

método3 ( )

…………………………

MétodoN ( )

Los objetos son los sustantivos dentro de las especificaciones del problema, y las acciones contenidas en las especificaciones se suelen corresponder con los métodos de esos objetos.

Por ejemplo si se deseara elaborar un programa que permita calcular el sueldo de los empleados de una empresa.

Debemos determinar cuántos empleados tiene. Para identificar cada empleado necesitamos: nombre, horas trabajadas y monto por hora trabaja. El monto a pagar al empleado se calcula como monto por hora multiplicado por la cantidad de horas trabajadas por empleado.

Desde la perspectiva de un sistema de información: entrada – proceso – salida.

Entrada: nombre, monto por hora, cantidad de horas.

Proceso: calcular el monto a pagar.

Salidas: nombre y monto a pagar.

En el problema planteado, el sustantivo es “empleado”, por lo tanto los objetos son personas. La representación gráfica se puede ver en la figura 3.

Figura 3

Nombre: Juan

Horas trabajadas: 30

Cuota hora: 50$

Sueldo: 150$

Objeto Empleado

Nombre: Pablo

Horas trabajadas: 50

Cuota hora: 70$

Sueldo: 350$

Objeto Empleado

En este caso tenemos una colección de objetos similares e independientes, son objetos de tipo persona.

La clase es una representación abstracta que describe a un conjunto de objetos; es decir es un tipo de dato abstracto que representa un conjunto de objetos que tienen en común una misma estructura (datos) y un mismo comportamiento (las mismas funciones), es decir son la misma cosa y hacen lo mismo.

Una vez identificados los objetos pasamos a determinar las clases que necesitamos para resolver el problema.

Cada instancia de una clase se representa por una línea punteada, la representación de la clase y sus instancias, se muestra en la figura 4

Ya expresamos que una de las características importantes de la POO es la encapsulación.

Un conjunto de objetos se representa en forma abstracta por una clase, este tipo de dato está compuesto por dos elementos: datos y métodos. La encapsulación significa que se colocan juntos los datos y los métodos dentro de un objeto; esto implica que quien programa debe pensar en el código y los datos juntos durante el diseño del programa.

CLASES

Las clases son como plantilla o modelos que describen como se construyen ciertos tipos de objetos. Cada vez que se crea un objeto de una clase, se crea una instancia de esa clase.

Una clase es una colección de objetos similares. Una clase puede tener muchas instancias y cada una es un objeto independiente.

C++ utiliza la palabra reservada class para crear clases.

La declaración de una clase es similar a la de una estructura. La sintaxis es:

class nombreclase

{ //miembro dato1

//miembro dato2

//miembro datoN

//función miembro 1

//función miembroN

}

Los miembros de una clase tienen especificadores de acceso o visibilidad. C++ ofrece 3 niveles de visibilidad para los miembros datos y miembros funciones, es son sección pública, sección privada y sección protected.

Existen dos funciones miembros especiales que pueden declararse en cada clase: los constructores y los destructores.

Los métodos constructores son los que permiten crear una variable de una clase e inicializarla a la vez. Las variables de clases de clases son relativamente complejas de crear e inicializar, los constructores de la clase automatizan el procedimiento de crear e inicializar las variables (atributos) de una clase. Un destructor es una función miembro que se invoca cuando se destruye la clase.

Todas las clases tienen un constructor y un destructor implícito, aunque no esté expresamente declarado en la clase.

Figura 4

Nombre: Juan

Horas trabajadas: 30

Cuota hora: 50$

Sueldo: 150$

Objeto Empleado

Nombre: Pablo

Horas trabajadas: 50

Cuota hora: 70$

Sueldo: 350$

Objeto Empleado

Empleado

VISIBILIDAD DE UNA CLASE

C++ ofrece 3 niveles distintos de visibilidad para los miembros de una clase: sección pública, sección privada y sección protegida. La visibilidad implica establecer como los usuarios acceden o interactúan con la clase.

Los alcances son:

* Sección pública (public): este es el nivel más amplio de visibilidad, los miembros que se encuentren en esta sección son visibles a los métodos miembros de la clase misma que los contiene, instancias de la clase, métodos miembros de clases descendientes y sus instancias. Si en particular el miembro público es una variable (atributo), los usuarios de la clase pueden acceder, sea para leer, sea para cambiar el contenido de dicha variable; y si el miembro público en cuestión es un método todos los usuarios de la clase pueden llamar a dicho método
* Sección privada (prívate): este nivel está más restringido que el acceso público. Los miembros que fueron declarados como privados solo podrán ser accedidos por los métodos de la propia clase que los contiene. Si una variable o un método está en la sección privada, ningún usuario podrá acceder a ellos, sólo los métodos de la propia clase pueden manipular variables o métodos privados y deben estar en la sección pública.
* Sección protegida (protected): Sólo los miembros de la propia clase y sus clases descendientes pueden acceder a los miembros que fueron declarados como protegidos. Este nivel de acceso es intermedio entre los otros dos.

El modo por defecto para la sección de una clase es privado. Esto significa que si no existe una etiqueta al principio de la definición de la clase, los miembros se consideran privados hasta que aparece una etiqueta.

class persona

{int edad; //atributo privado

public:

void carga\_edad ( int ): // método público

protected:

int dato; // sección protegida

void ingresar\_dato ( int );

};

Tener presente que las secciones pueden aparecer en cualquier orden, cada sección puede aparecer más de una vez y como ya lo expresamos antes, si no se especifica sección el compilador de C++ considera a los miembros como privados.

CONSTRUCTORES

Es un tipo de método miembro especial que se coloca en la sección pública. Los métodos constructores se utilizan para “construir la zona o espacio de memoria” que contendrá los atributos del objeto e inicializarlos al mismo tiempo. Entonces el constructor permite crear una instancia de una clase (objeto) o variable de una clase e iniciarla a la vez.

El constructor es llamado automáticamente por el compilador de C++ siempre que una clase se instancia. Es más puede que el programador no cree un método constructor, en este caso el compilador creará uno por defecto, e inicializará los números enteros a 0, los punteros a null, los reales a 0.0.

En C++ se utilizan las siguientes reglas para los constructores:

* El nombre del constructor es el mismo de la clase
* Una clase puede tener cualquier número de métodos constructores, e incluso ninguno
* Un constructor se llama una sola vez, cuando se declara una instancia de la clase (objeto) o la variable de tipo objeto. En este momento se inicializan los atributos del objeto que representa la clase en cuestión.
* Los métodos constructores pueden tener o no tener parámetros, e incluso pueden tener parámetros fijados a valores por defecto
* Los métodos constructores no tienen tipo asociado, ni siquiera tipo void
* Existen 3 tipos básicos de constructores:
* De copia
* Por defecto
* Constructor ordinario

El compilador es el encargado de ejecutar el constructor apropiado, cuando hay varios tipos de métodos constructores, fundamentalmente va a depender de cómo se instancie la clase. Cada declaración de una clase implica la ejecución de un constructor

Algunos ejemplos de instancias de clases:

………// include

class persona {

public:

persona ( ) ; // constructor por defecto

persona ( int t, doble m); // constructor ordinario pasa parámetros

persona ( persona & n); // constructor de copia

…….

};

…….

int main ( )

class persona p; // se invoca al constructor por defecto

class persona p2 ( 3, 4.5); // constructor parametrizado

class persona p3 ( p2); // utiliza el constructor de copia

Un método constructor de copia o copiador se utiliza para copiar un objeto a otro, es decir el método constructor de copia permite crear una instancia de una clase utilizando una instancia ya existente (del mismo tipo). Este tipo de constructor tiene un único parámetro por referencia a una instancia de la misma clase y copia de los miembros datos de un objeto a otro

Un constructor por defecto es un constructor que no pasa parámetros, o que pasa parámetros con argumentos por defecto.

Los constructores por defecto son útiles cuando se desea que implícitamente el compilador los inicialice o cuando por ejemplo se quiere inicializar un arreglo o un puntero de tipo char (cadena).

Por ejmplo

persona : : persona ( void)

persona : : persona ()

persona : : persona ( int edad =10)

Si se desea cambiar el estado de un objeto después que fue instanciado se deberá utilizar un método contenido en la clase para ello, y no el constructor.

DESTRUCTOR

Un destructor es un método miembro que se invoca cuando se destruye una clase. Todo clase tiene un destructor implícito que lo suministra el compilador. Es decir el programador puede no incluir en el código de la clase un destructor, y en este caso el campolador ejecuta uno por defecto.

En C++ se utilizan las siguientes reglas para los destructores:

* El nombre del destructor es el mismo de la clase y debe ir precedido por el carácter tilde ( ~).
* Una clase sólo puede tener un método constructor, o ninguno.
* Un destructor se llama automáticamente, siempre que una variable de ese tipo (objeto) sale fuera de ámbito.
* El método destructor no puede tener parámetros.
* El método destructor no tiene tipo asociado, ni siquiera tipo void.

Ejemplo

:………// include

class persona {

public:

persona ( ) ; // constructor por defecto

//métodos necesarios

………..

~persona ( );

};

persona::~persona ( )

{ cout << “ el objeto se destruyo”<< endl,}

…….

int main ( )

class persona p; // se invoca al constructor por defecto

return 0;

}

El output en pantalla será.

el objeto se destruyo

Apreciar que el método destructor no se invocó implícitamente.

FUNCIONES AMIGAS

El encapsulamiento y el principio de ocultamiento de la información , en POO implica que solo los métodos propios de una clase pueden acceder a los miembros privados de una clase. A veces por alguna circunstancia es necesario que funciones externas a la clase accedan a miembros privados de una clase; esta posibilidad se realiza por medio de funciones amigas que de alguna manera vienen a romper el paradigma en menor medida que se los atributos privados se declarasen como públicos.

Las funciones amigas NO pertenecen a la clase. Estas funciones ordinarias tienen acceso a todos los datos privados y protegidos de una o más clases.

Para que C++ reconozca a la función como amiga hay que declararla en la clase que sea ser amiga precediendo al prototipo de la función con la palabra reservada friend.

La sintaxis es:

friend prototipo\_ de\_función;

La definición friend puede ir en cualquier parte de una clase, aunque habitualmente se colocan todas juntas al comienzo de la clase

class datos

{

friend int comparar ( int, datos);

friend void establecer ( char, datos);

public: // ….

private:

int m

//…..};

// fuera de la clase se define la función amiga

int comparar ( int t, datos r)

{ return r.m == t);}

La función camparar no pertenece a la clase es una función ordinaria excepto que accede al dato privado m.

CLASES AMIGAS

C++ también permite declarar a todas o a algunas las funciones miembros de una clase amigas de otra clase.

class b;

class a

{ friend class b; // todas las funciones de la clase b son amigas de la clase a y no al revés

// código}

Para crear acceso recíproco hay que declarar como friend en ambas clases, por ejemplo

class b;

class a

{ friend class b; // todas las funciones de la clase b son amigas de la clase a

// código}

class b

{ friend class a; // todas las funciones de la clase a son amigas de la clase b

// código}

También es posible que algunas funciones de una clase sean definidas amigas, sin necesidad de hacer a todas las funciones miembros como amigas.

class B

{ void funcion1( );

int funcion2 ( ) ;

// código}

class N

{ friend void funcion4 : : funcion1 // únicamente la funcion1 tiene acceso total a N

// código}

Este último código nos muestra como una función declarada amiga de una clase tiene acceso a todos los datos de la clase, sin ser relevante la sección en la que fue declarada amiga

HERENCIA Y POLIMORFISMO

Una de las características de la POO es la herencia y el polimorfismo.

La herencia permite rehusar código en clases descendientes.. Una clase descendiente hereda los miembros de su o sus clases ascendentes. Cuando una clase se hereda de otra, la clase original se llama clase base o clase padre y la nueva clase se llama clase hija o simplemente clase derivada. Un sistema de clases creado utilizando herencia se llama jerarquía de clases.

Cuando se realiza herencia, la clase derivada puede heredar las características de la clase padre con la posibilidad de usarlas tal como fueron definidas en el padre o puede cambiarlas, eliminarlas o incluso ampliarlas.

La sintaxis es:

clase D : modificador\_acceso B

esta línea de código está diciendo que la clase D hereda con acceso….. de la clase B ( padre)

Si se omite el modificador de acceso, entonces por defecto es privado.

La clase hija puede agregar atributos (miembros dato) para acentuar la especificidad de la nueva clase, puede agregar métodos. Los únicos métodos que no se heredan son los constructores, el destructor y cualquier operador de asignación sobrecargado (operador = ( ) ), lo mismo ocurre con con las relaciones de amistad. ( métodos friend).

A continuación se muestra una tabla con la accesibilidad de las clases derivadas

|  |  |  |  |
| --- | --- | --- | --- |
| Modo de  derivación | Estado del miembro en la clase base | Estado del miembro en la clase derivada |  |
|  |  |  |  |
| private | private  protected  public | inaccesible  private  private |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| protected | private  protected  public | inaccesible  protected  protected |  |
|  |  |  |  |
|  |  |  |  |
| public | private  protected  public | inaccesible  protected  public |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

HERENCIA MULTIPLE

La herencia simple implica que una clase tiene solamente una clase base. La herencia múltiple puede tener dos o más clases bases.

clase a: modificador\_de\_acceso base1, ……, modificador\_de\_acceso base2

POLIMORFISMO

El polimorfismo es una característica de la POO, por la cual métodos con el mismo nombre y signatura se comportan de modo diferente según el momento en que se invocan o según la clase a la que pertenecen. Para que pueda haber polimorfismo, debe haber herencia. No puede implementarse la característica de polimorfismo, si no se realiza herencia.

C++ implementa el polimorfismo mediante el uso de funciones virtuales a través del uso de la palabra reservada y de la “ligadura dinámica”. Entonces una función declarada como virtual en la clase base, y en alguna de las clases derivadas redefinida, puede actuar polimórficamente en esa clase derivada.

Cuando se invoca una función virtual, para el lenguaje C++, el método invocado se determina (cuál de ellos, puesto que al menos hay dos con idéntico nombre y firma) en el momento de la ejecución (a diferencia de C, para el cual la función se determina en tiempo de compilación).

La ligadura dinámica, implica que la dirección de memoria de la función polimórfica se determina mientras está corriendo el programa, es decir en tiempo de ejecución. La ligadura dinámica solo se da si hay jerarquía de clases en un programa. La contrapartida a esta flexibilidad del lenguaje es el tiempo de procesado, es decir disminuye la velocidad de ejecución.

Para crear funciones virtuales, hay que anteponer al tipo de retorno del método la palabra reservada virtual. Una vez declarada una función como virtual, solo puede anularse ésta en clases descendientes con funciones virtuales en las clases hijas, además las funciones virtuales, pueden anular funciones no virtuales en clases ascendentes.

C++ sólo admite como funciones virtuales, las funciones miembros de una clase, es decir las funciones ordinarias o funciones amigas NO pueden ser virtuales.

Ejemplo

//include

class A

{ public:

virtual double fun1 ( double z){ return z\*z;}

double fun2 ( ){ return 2 \* fun1 ( z) ;}

private:

int dato;

double info;

};

class B: public A

{ public:

B ( ) ;// constructor

virtual double fun1 ( double z){ return z \* z \*z;}

int fun6 ( ){ //código }

private:

double info1;

};

…….

int main ( )

{ class B b1;

cout << “ligadura dinámica con función polimórfica” << b1.fun2( 2.0);<<endl;

return 0;

}

En este ejemplo, es importante notar como el objeto b1 de tipo B invoca al método fun2 que no está escrito (por decirlo) en su clase, pero que por medio de la herencia simple tiene acceso (para el compilador la clase B tiene en total 3 métodos) a él. Al ejecutarse el método fun2, en tiempo de ejecución el compilador llama al método fun1 de la clase B y anula el método de igual nombre y parámetros (idéntica signatura) de la clase A.

CLASES ABSTRACTAS

C++ también permite crear clases que sólo se pueden usar como clases bases, es decir no se pueden instanciar objetos de ellas. Estas clases abstractas, se utilizan para definir conceptos abstractos (de allí el nombre) sin dar detalles específicos de cómo cumplirá su objetivo. Por ejemplo se puede crear como abstracta la clase persona, y a su vez está utilizarla para derivar otras clases como: estudiante, docente, empleado, y otras que tengan el concepto genérico de persona.

Normalmente la clase abstracta está en la raíz de la jerarquía de clases, aunque en una misma jerarquía pueden haber más de una clase abstracta.

Para crear una clase abstracta, C++ necesita que al menos una función virtual de una clase padre sea virtual pura. Una función virtual pura es aquella que no tiene código o cuerpo para lo cual hay que inicializarla a cero, es decir de la siguiente forma:

virtual tipo\_retorno nombreFuncion ( lista\_parametros) = 0;

por ejemplo :

virtual int fun1 ( int t, char a) = 0; // esta es una función virtual pura

**Observar que la función fun1 no tiene cuerpo y no es una función virtual nula**

Una función virtual nula es la siguiente:

virtual void Funcion ( ){ };//función virtual nula

Para visualizar mejor la funcionalidad de las clases abstractas, se proporciona el siguiente ejemplo: “se creará una clase figura como padre y una clase hija rectángulo”

//include necesarios y adecuados

class Figura

{ public:

virtual double area ( ) =0; // function virtual pura

virtual double permetro ( ) = 0 ;// function virtual pura

………….

private:

……..

};

class Rect : public Figura

{ public:

Rect ( ) ;// constructor

double perimetro ( ){ return (2 \*base + 2 \*altura;}

virtual double area ( ){ return base \* altura; }

……………………

private:

double base,altura;

};

…….

La clase Figura es abstracta ya que posee más de un método virtual pura, estos son: área y perímetro

En el ejemplo la función perímetro se implementa en la clase derivada, pero como no se la declaro virtual no podrá ser reemplazada por otra función con el mismo nombre y signatura en clases derivadas de Rect . El caso es distinto para la función área de la misma clase Rect, como este método área (implementado en la clase derivada Rect) es virtual, las clases hijas de Rect, podrán redefinir este método área.

EJERCICIOS DE OBJETOS

1. Crear una clase punto que permita representar un punto como objeto en el espacio bidimensional. La clase tendrá un constructor para inicializarlo a cero (0,0). La clase tendrá los siguientes métodos:

* Ingresar valores desde el teclado para crear el punto.
* Mostrar las coordenadas del punto por pantalla.
* Calcular y mostrar la distancia del punto al origen del sistema de coordenadas cartesianas.

En el main se creará una instancia de la clase. El ingreso de valores y de información a mostrar en pantalla se hará desde el main.

1. Crear una clase vector, la cual representará un arreglo estático de 10 elementos de tipo entero. Un constructor por defecto iniciará el arreglo a 0. La clase tendrá métodos para:

* Cargar datos en el arreglo.
* Mostrar en pantalla los elementos almacenados en el arreglo.
* Determinar el mayor y el menor elemento del arreglo junto con sus posiciones. Esta información se mostrará en pantalla en el main.

1. Crear una clase vector, la cual representará un arreglo estático de 10 elementos de tipo entero. Un constructor por defecto iniciará el arreglo a 0. La clase tendrá métodos para:

* Cargar datos en el arreglo.
* Mostrar en pantalla los elementos almacenados en el arreglo.
* Determinar si está ordenado de menor a mayor. Esta información se mostrará en pantalla en el main.
* Ordenar de menor a mayor

Implementar el main, para un objeto

1. Crear una clase Matriz, la cual representará una matriz de 5x5 de elementos de tipo entero. Un constructor por defecto iniciará la matriz a 0. La clase tendrá métodos para:

* Cargar datos en la matriz.
* Mostrar en pantalla los elementos almacenados en la matriz.
* Determinar el promedio simple de información almacenada en el cuadro. Esta información se mostrará en pantalla en el main.

Implementar el main con una instancia de la clase.

1. Crear una clase Pila, la cual permite administrar una estructura dinámica de tipo pila de números enteros. La clase tendrá métodos para:

* Insertar un elemento en la pila
* Borrar un nodo
* Recorrerlo y mostrar la información por pantalla.

1. Crear una clase Cola, la cual permite administrar una estructura dinámica de tipo cola de números enteros. La clase tendrá métodos para:

* Insertar un elemento en la cola
* Borrar un nodo
* Recorrerlo y mostrar la información por pantalla.

1. Crear una clase Persona. Los atributos son: nombre (cadena), edad (entero), sexo (codificado, entero), número de documento de identidad (unsigned long int).

Generar un arreglo de tipo persona de 10 elementos. Desde el main se determinará cuantos son varones y cuantas son mujeres.

1. Implementar una clase que permita generar un número random, utilizando el método rand( ) de la librería de matemática. Utilizar constructor, destructor y los métodos necesarios para generar el número aleatorio encapsulado en el objeto, posteriormente mostrarlo por pantalla.
2. Generar la clase numero-aleatorio, cuyos atributos son un número unsigned int, y dos miembros int que constituyen el límite inferior y el límite superior del intervalo dentro del cual debe generarse el número aleatorio. Incluir los métodos necesario para guardar el número generado en el objeto y para mostrarlo por pantalla.
3. Crear una clase vector, la cual representará un arreglo estático de tamaño N elementos de tipo entero. Un método, mediante la función rand ( ) generará el tamaño aleatoriamente del arreglo entre 1 y 20. Otro método generará los elementos del arreglo aleatoriamente mediante la función rand ( ). El arreglo no tendrá repeticiones de valores. La clase incluirá un constructor y un destructor, además de un método para mostrar los elementos del arreglo por pantalla.
4. Crear una clase vector, la cual representará un arreglo de N elementos de tipo entero. Un constructor por defecto iniciará el arreglo a 0. La clase tendrá métodos para:

* Cargar N datos en el arreglo comprendidos entre 1 y 100, se admiten repeticiones de elementos. Se podrá cortar el ingreso de datos cuando el usuario ingrese 0( sin haber alcanzado el tamaño N)
* Mostrar en pantalla los elementos almacenados en el arreglo.
* Determinar el mayor elemento almacenado en el arreglo y su posición. Esta información se mostrará en pantalla en el main.
* Determinar los elementos que se repiten y el número de ocurrencia de los mismos.

Implementar el main, para dos instancias de la clase.

1. Crear una clase Matriz, la cual representará una matriz de 4x4 de elementos de tipo entero. Un constructor por defecto iniciará la matriz a 0. La clase tendrá métodos para:

* Cargar datos en la matriz.
* Mostrar en pantalla los elementos almacenados en la matriz.
* Determinar el promedio simple de información almacenada en el cuadro. Esta información se mostrará en pantalla en el main.

Crear una clase vector, la cual representará un arreglo de 10 elementos

de tipo entero. El arreglo tendrá los siguientes métodos:

* Cargar datos en el arreglo.
* Mostrar en pantalla los elementos almacenados en el arreglo.
* Determinar el promedio simple de información almacenada en el arreglo. Esta información se mostrará en pantalla en el main.

Implementar el main una instancia de cada clase, determinando cual de los dos objetos tienen el promedio simple más alto. Usar una función amiga.

EJERCICIOS DE HERENCIA

1. Crear una clase madre llamada Punto, con constructor para inicializar y tomar valores desde el teclado. La clase deberá tener métodos para mostrar por pantalla la información del objeto. Crear una clase circunferencia hija con acceso público a la clase Punto. La clase Circunferencia representa una circunferencia mediante un punto (su centro) y cuyo radio se ingresa como parámetro a través de su constructor.
2. Crear la clase Persona y la clase hija Empleado. Los atributos de la clase Persona son: nombre, apellido, sexo, edad y estado civil. Tiene todos los métodos para setear información y para mostrar por pantalla los atributos (métodos de getter). Los atributos de la clase Empleado son: categoría, antigüedad y remuneración. También dispone de todos los métodos necesarios (setter y getter) para introducir datos y para mostrar la información por pantalla. Implementar el main para 5 empleados. Se desea realizar las siguientes consultas:
3. Listar el personal por antigüedad
4. Listar el personal por remuneración
5. Listar el personal femenino de categoría superior o igual a la media.
6. Crear una clase madre llamada Coordenada, en esta clase un método rand() genera al azar una coordenada .Crear otra clase llamada complejo, la cual permitirá representar un numero complejo ( z = x + j y). Utilizar herencia para que la parte real y la parte imaginaria sea generada por la clase Coordenada. Las clases madre e hija tendrán todos los métodos establecer (setter) y obtener (getter) para generar los objetos y mostrar su estado por pantalla.
7. Crear la clase Persona y la clase hija Alumno. Los atributos de la clase Persona son: nombre, apellido, edad y sexo. Tiene todos los métodos para establecer (setter) y para obtener (getter) información de los objetos. Los atributos de la clase Alumno son: promedio general, número de inasistencias, legajo. También dispone de todos los métodos para establecer y obtener la información necesaria (setter y getter). Escriba un programa que cree un arreglo de 20 elementos de tipo alumno, se deberá listar:
8. Listar los legajos de los alumnos con más de 3 inasistencias.
9. Listar los apellidos de los alumnos sobresalientes (alumnos con promedio superior a 7)
10. Listar los varones mayores a la media de la edad y mayores a la media del promedio general de la población.
11. Crear la clase Empleado, como clase madre. Como clases hijas crear: EmpleadoAsalariado, el cual tiene un salario semanal; un EmpleadoPorHora debe tener un sueldo y el número de horas trabajadas; un EmpleadoPorComision debe tener una tasa de comisiones y ventas brutas. Cada clase debe tener los constructores apropiados, los métodos establecer y obtener. Escriba un programa que cree instancias de objetos de cada una de las clases, y que muestre toda la información asociada con cada objeto (incluyendo la información heredada).
12. Escriba una jerarquía de herencia para las clases Cuadrilátero, Trapeziode, Paralelogramo, Rectángulo y Cuadrado. Use Cuadrilatero como la superclase de la jerarquía. Haga la jerarquía de todos los niveles que sea posible. Especifique las variables de instancias y los métodos para cada clase. Los datos private de Cuadrilatero deben ser los pares de coordenadas x-y para las cuatro esquinas del Cuadrilatero. Escriba un programa para instanciar objetos de sus clases y que muestre el área de cada uno de los objetos derivados (excepto Cuadrilatero).
13. Crear las clases mesa, círculo y cuadrado. Utilizar los atributos y métodos necesarios para delimitar claramente estos objetos. Mediante herencia múltiple crear objetos MesaRedonda y MesaCuadrada.

Atributos de mesa pueden ser: color, material, número de patas/pilares.

Atributos de objeto cuadrado: perímetro y área.

Atributos de objeto círculo: perímetro y área.t

1. Crear las clases: Empleado y profesión, como clases madres. Como clases hijas crear: IngenieroEmpleadoAsalariado, el cual tiene un salario mensual y categoría; un TécnicoEmpleadoPorHora tiene como atributo específicos: categoría, remuneración la cual se paga conociendo el monto por hora, horas trabajadas por día, días trabajos al mes; un AdministrativoEmpleadoPorComision, cuyos atributos específicos son: tasa de comisiones, ventas brutas y sector ventas. Cada clase debe tener los constructores apropiados, los métodos necesarios para ingresar datos y obtener información de los objetos. Atributos de la clase Empleado: nombre y edad. Atributos clase profesión: tipo\_profesión, categoría

Escriba un programa que cree instancias de objetos de cada una de las clases hijas, y que muestre toda la información asociada con cada objeto (incluyendo la información heredada).

1. Crear 3 clases llamadas: Proceso con los atributos: tiempo\_proceso, número\_fallas, otra llamada Corte con los atributos forma\_corte, potencia-corte, y una última clase llamada Enfriado con los atributos: tipo\_aireado, presión\_aireado .

Se pide.

1. Crear los objetos ProcesoCorte y ProcesoAireado
2. Una función amiga a las clases hijas comparará los números de fallas de ProcesoCorte y ProcesoAireado.
3. Implementar el main para instanciar las clases del apartado a), mostrar por pantalla el resultado de la comparación de la función amiga.

EJERCICIOS DE POLIMORFISMO

1. Crear la clase elemento\_grafico, la cual tiene como hijos línea y cuadrado. A su vez la de clase línea hereda línea\_horizontal y línea\_vertical. El método pintar de cada objeto consiste en la realización grafica de una figura acorde al objeto por ejemplo la línea se podrá pintar por pantalla con el elemento “-“ que se repite un número de veces para darle la extensión de la línea. El método pintar es polimórfico
2. Crear una clase base llamada vehículo con los atributos: año del vehículo y costo, métodos para obtener sus valores y un método virtual llamado mostrar por pantalla. De esta clase se derivarán otras dos llamadas vehículo\_aéreo y vehículo\_terrestre, con sus atributos horas\_vuelo y kilometraje respectivamente. Además deberán incluir métodos para obtener y establecer esos datos junto con la implementación del método virtual mostrar por pantalla. En el main crear objetos de las clases derivadas
3. Crear las clases: Empleado y profesión, como clases madres. Como clases hijas crear: IngenieroEmpleadoAsalariado, el cual tiene un salario mensual y categoría; un TécnicoEmpleadoPorHora tiene como atributo específicos: categoría, remuneración la cual se paga conociendo el monto por hora, horas trabajadas por día, días trabajos al mes. Cada clase debe tener los métodos necesarios para ingresar datos y obtener información de los objetos. Atributos de la clase Empleado: nombre y edad. Atributos clase Profesión: tipo\_profesión, categoría

Escriba un programa que cree instancias de objetos de cada una de las clases hijas, y que muestre toda la información asociada con cada objeto (incluyendo la información heredada).Crear un método polimórfico que pague a los empleados, cuando corresponde al mes de su cumpleaños 500$ con categoría menos a 4 y 700$ con categoría igual o superior a 4.

1. Crear la clase figura como clase abstracta con los miembros necesarios para que puedan crearse las clases hijas rectángulo y triángulo. En el main crear un objeto de cada clase hija.
2. Crear la clase vehículo como clase abstracta con los miembros necesarios para que puedan crearse las clases hijas avión y auto. En el main crear un objeto de cada clase hija.