Algoritmos

Programación Orientada a Objetos

Logro de sesión

Al finalizar la sesión, el estudiante aplica los conceptos de clases y objetos en la construcción de programas.

Semana 2: Clases y Objetos

Contenido:

- Programación Estructurada
- Programación Orientada a Objetos
  - Conceptos
  - Ventajas y Desventajas
  - Principios Fundamentales
  - Clases y Objetos.
  - Atributos, Métodos Mensajes.

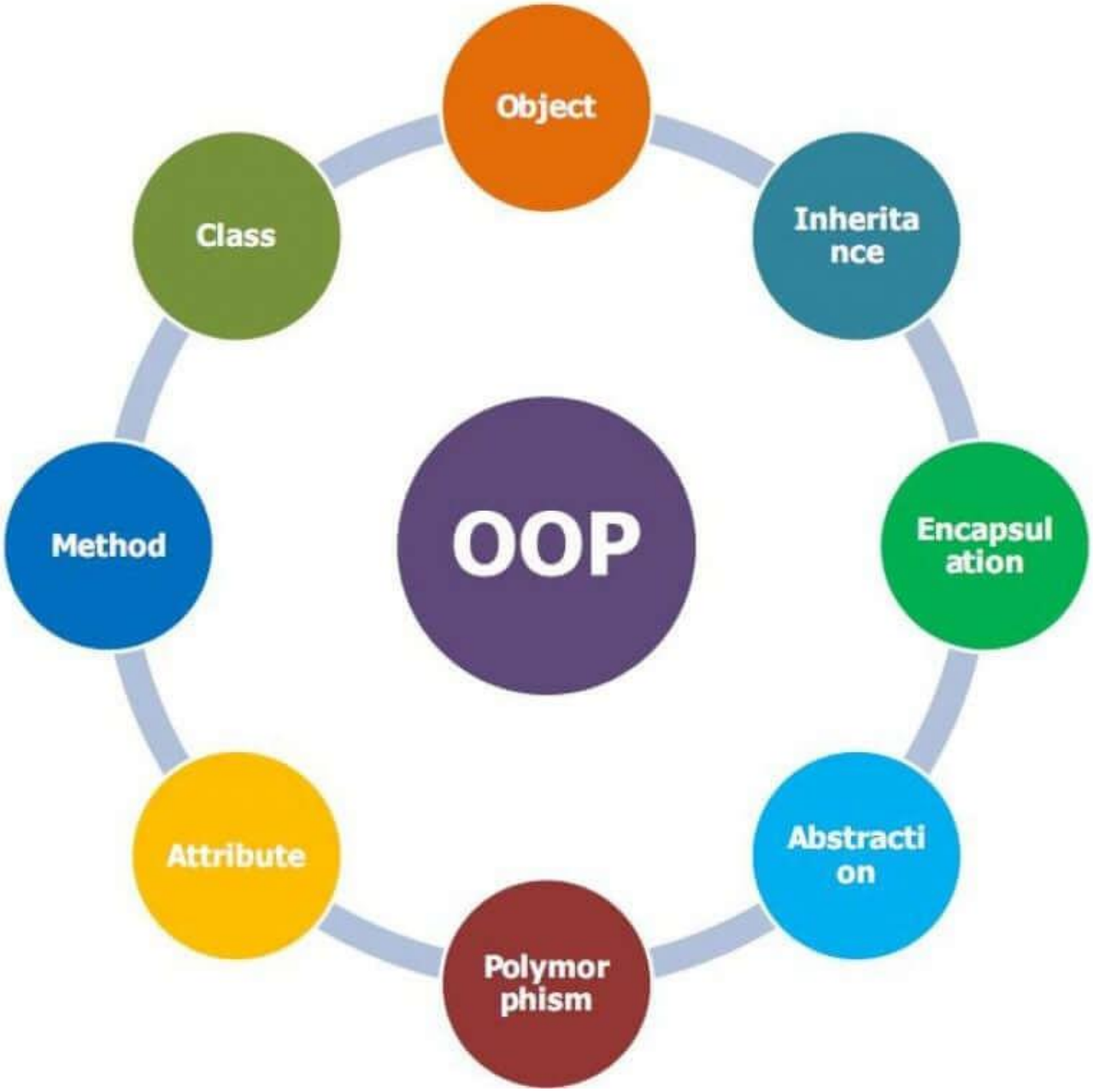
Programación Estructurada

- La programación estructurada establece un orden particular.
- Maneja por separado al código y a las estructuras de datos
- Los algoritmos trabajan sobre las estructuras de control.
- Este tipo de programación da mayor importancia al código que a las estructuras de datos.

Algoritmos + Estructuras de Datos = Programas

POO

Programación Orientada a Objetos



¿Qué es POO?

Es un paradigma de programación en la que los **objetos** se utilizan como metáfora para emular las entidades reales del negocio a modelar.

En el paradigma POO se trabaja con "**objetos**" que intercambian información entre sí, pero conservando cada uno un estado y unos datos que les son propios y que no son visibles desde otros objetos

Es un paradigma de programación, un estilo y una forma de pensar para la solución de diferentes problemas, este paradigma nuestras aplicaciones están basadas en **objetos** en lugar de una serie de comandos y en datos en lugar de la lógica

Programación Orientada a Objetos

La **POO** fue concebida por quienes reconocían un mundo poblado de ***objetos que interactúan*** entre si de acuerdo a su ***propia naturaleza***.

Programación Orientada a Objetos

- ✓ Cada objeto tiene su **propia naturaleza**. La propiedad o característica de un objeto lo distingue de otro.
- ✓ Cada objeto existe para **un propósito**. Este propósito define las **acciones**, los **procedimientos**, **servicios** o responsabilidad que un objeto puede proporcionar.
- ✓ Los **servicios** solicitados a los objetos depende de la naturaleza de los mismos, por ejemplo:
  - ❑ No se puede ***conducir*** un ***foco***
  - ❑ No se puede hacer ***volar*** un ***interruptor***.

Estas acciones o servicios mostrados son inapropiados porque **NO** forman parte del comportamiento natural de los objetos.

POO: Ventajas / Desventajas

- Ventajas
  - ✓ Reusabilidad.
  - ✓ Extensibilidad.
  - ✓ Facilidad de mantenimiento.
  - ✓ Portabilidad.
  - ✓ Rapidez de Desarrollo.
  - ✓ Más fáciles de entender porque se utilizan abstracciones más cercanas a la realidad.
- Desventajas
  - ✓ Curvas de aprendizaje largas
  - ✓ Dificultad en la abstracción

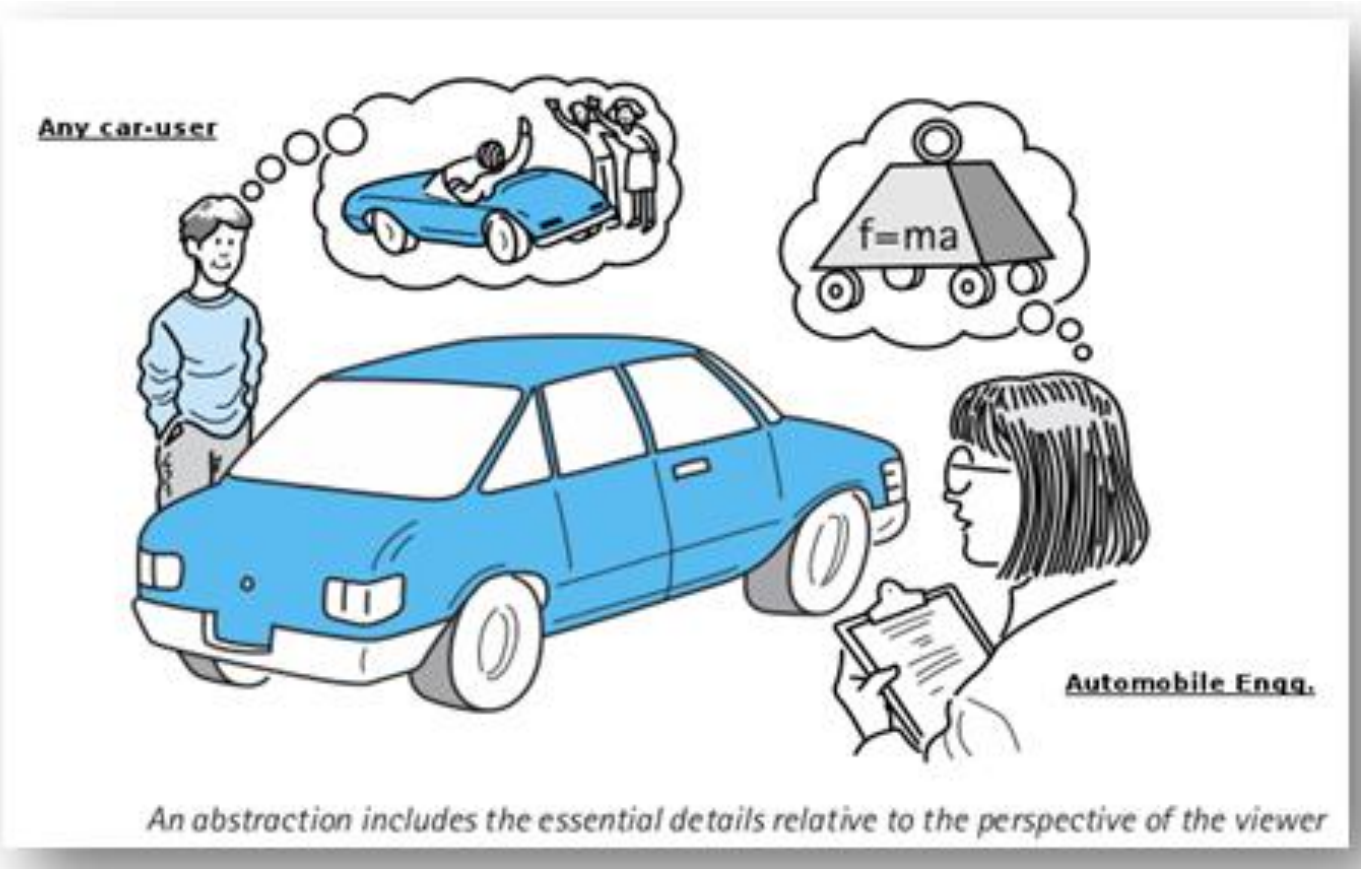
Principios Fundamentales de la POO

Principios Fundamentales de la POO

La abstracción consiste en seleccionar datos de un conjunto más grande para mostrar solo los detalles relevantes del objeto

Se encarga de identificar las características esenciales de un objeto, para capturar su comportamiento.

Principio #1: Abstracción



- ✓ Consiste en representar las características esenciales de un objeto, dejando de lado otras no esenciales.
- ✓ Una CLASE ES UNA ABSTRACCIÓN de un grupo de objetos, que tienen la posibilidad de realizar una serie de operaciones.

- ✓ Definir una abstracción significa describir una entidad del mundo real, no importa lo compleja que sea y luego utilizarla en la descripción de un programa.

Principio de Abstracción: Aplicación

CARACTERIZANDO EL OBJETO ESFERA

- ¿Una Esfera es un Objeto?
- ¿Qué caracteriza a una Esfera?
- ¿Qué Operaciones o cálculos se pueden hacer con una Esfera?

Principio de Abstracción: Aplicación

CARACTERIZANDO EL OBJETO ESFERA

¿Qué conozco de la Esfera, Que caracteriza una Esfera?

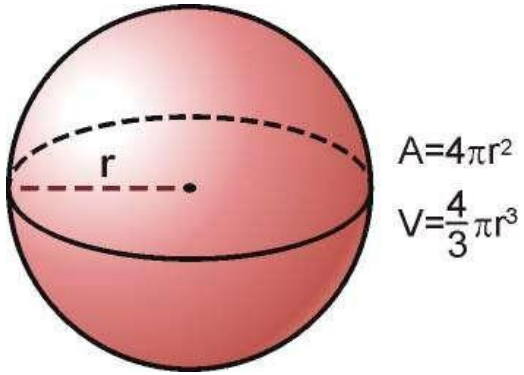
- radio

→ **Atributos**

¿Qué cálculos se puede hacer la Esfera?

- Calcular el área
- Cal cular el volumen

→ **Métodos**

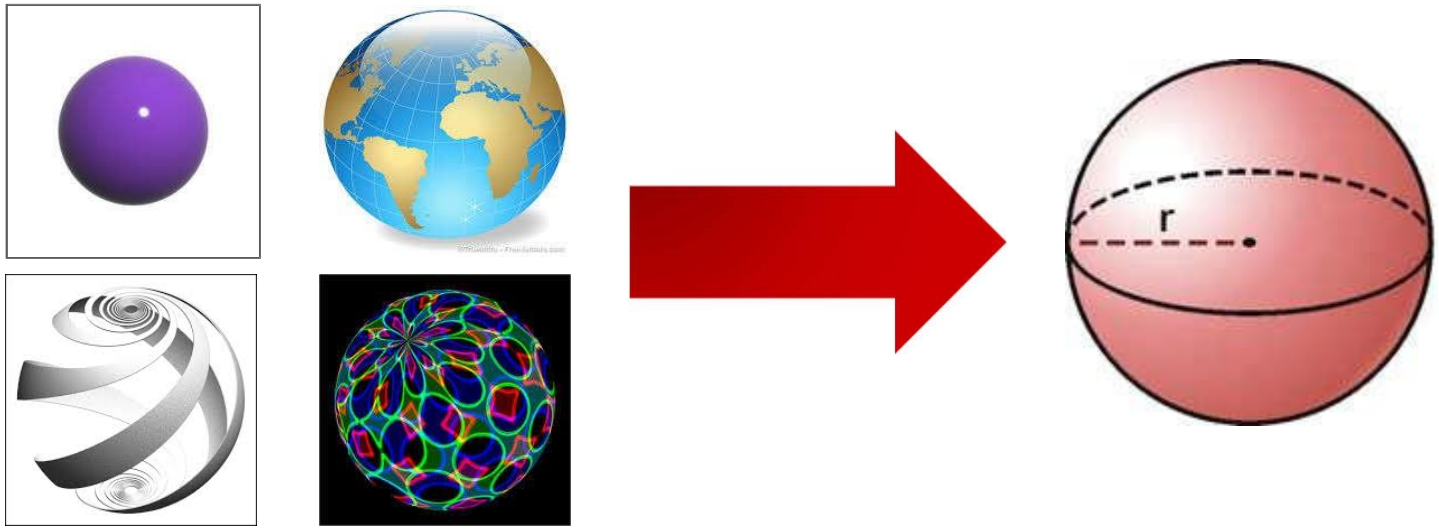


¿Qué otros cálculos u operaciones pueden hacerse?

- Reconocer el valor del radio
- Modificar el valor del radio

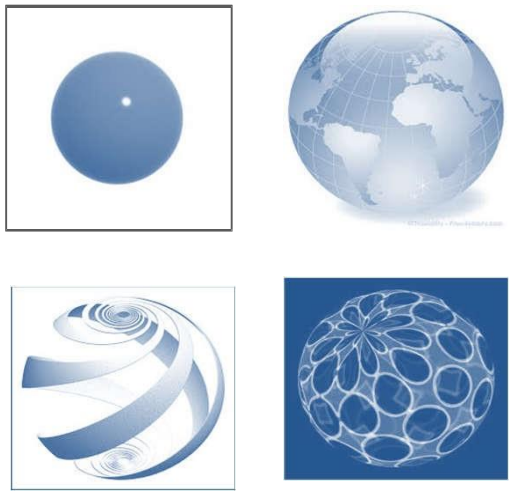
Clase & Objetos

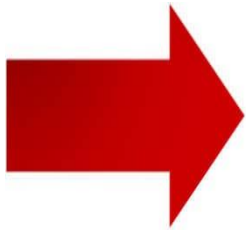
- Una **CLASE** es un nivel superior de ***abstracción*** que se corresponde con un conjunto de objetos que poseen las mismas propiedades y comportamientos.
- Una **CLASE** denota a una ***colección de objeto***s de un ***mismo tipo***.



Clase & Objetos

- La **CLASE** posee ***atributos y operaciones***. Los atributos son variables que tiene un tipo de dato asociado.
- Una **CLASE** se ***formaliza/representa*** gráficamente mediante un ***Diagrama*** que representa a una clase





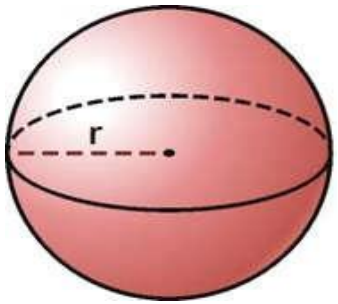
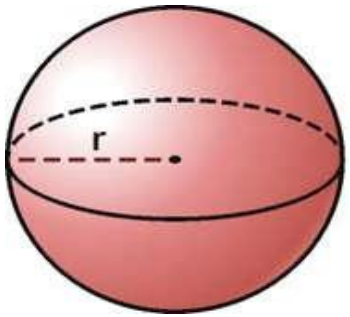


Diagrama de
La clase Esfera

Esfera
radio: int
area(): double **volumen():double**

Clases & Objetos

- El diagrama de una CLASE tiene tres bloque que denotan a la clase, las propiedades y métodos.



Nombre de la clase

Esfera

radio: int

area(): double
volumen():double

Variables Miembro
Atributos
Campos
Estados
Propiedades
Variables
Datos

Funciones Miembro
Métodos
Responsabilidades
Operaciones Servicios
Habilidades
Comportamiento
Procedimientos/Funciones

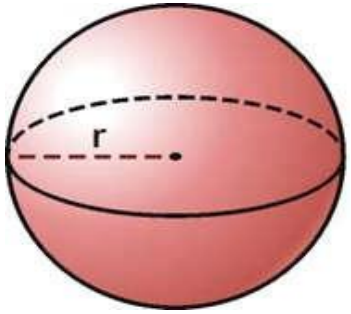
CLASE

PROPIEDADES
METODOS

Otros nombres usados para denotar cada bloque del diagrama

Clases & Objetos

- Una CLASE una vez formalizada, puede empezar a codificarse. Existe una relación directa entre el diagrama y el código de implementación de la clase.



Esfera
radio: int
area(): double volumen():double

```
class Esfera{
     //propiedades
     int radio;
     //métodos
     double area(){***}
     double volumen(){***}

}
```

Clases & Objetos

- Un **OBJETO** es un ejemplar de un concepto del mundo real que puede ser ***modelado*** por una **clase**.
- Un **OBJETO** posee las siguientes características:
  - **Tiempo de Vida:** Esta dada por la duración de un objeto en un programa. Los objetos son creados mediante un mecanismo denominado *instanciación*, y cuando dejan de existir se dice que son *destruidos.*
  - **Estado:** definido por sus atributos (el valor de cada uno de sus atributos)
  - **Comportamiento:** Todo objeto ha de presentar una interfaz, definida por sus *métodos*, para que el resto de objetos que componen los programas puedan interactuar con él.

Clases & Objetos

- Un **OBJETO** tiene un identificador que nunca cambia, comportamientos y atributos que son específicos para esa clase, pero cada objeto tiene sus propios valores para cada uno de sus atributos.

Clase: Punto

Punto
x: int y:int color:int
crear(): double mostrar():double mover():void ocultar():void

Objetos o Instancias de la Clase Punto

Objeto1

Atributos:
21
45
verde

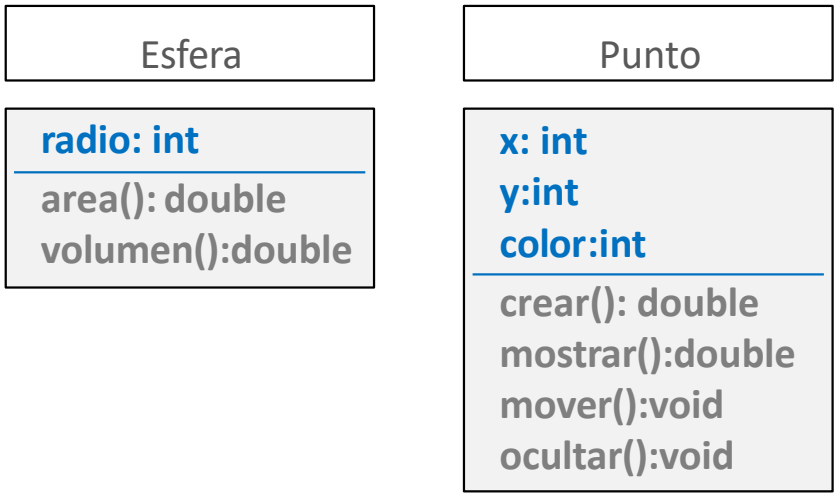
Objeto2

Atributos:
200
15
rojo

Atributos y Estado

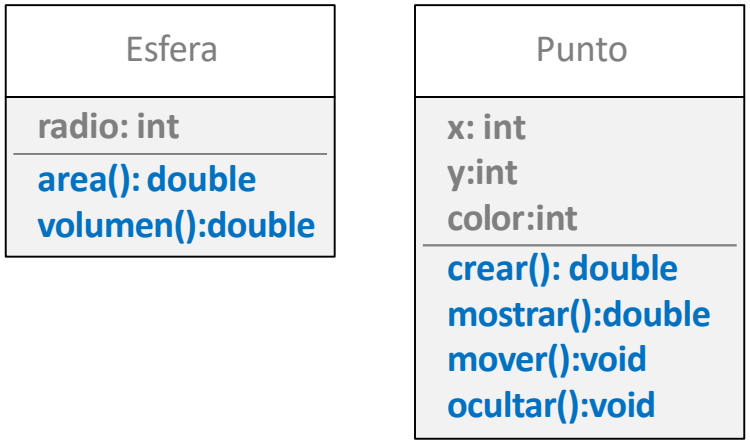
- Los **ATRIBUTOS** son las **características *o* propiedades** que manifiestan todos los objetos de una clase. Se utilizan para describir, identificar o informar el **estado**.

- El **ESTADO** de un objeto o clase viene dado por los **valores de sus ATRIBUTOS** en un instante dado.



Métodos

- Son ***procedimientos*** o ***funciones***.
- Son las ***acciones*** que deben ser ***realizadas por un objeto*** de una clase.
- Son las ***responsabilidades del objeto*** que incluyen tanto el **comportamiento** como el acceso a los **atributos** de los objetos y clases.
- Son los ***algoritmos*** que procesan los datos o atributos de un objeto y están contenidos en un bloque de código.



Método: Constructor/Destructor

- El **CONSTRUCTOR** es un método especial de la clase que se aplica automáticamente a los objetos en el momento de su creación.
- Propósito y Características:
  - Se utiliza para **inicializar** los **atributos** de la clase.
  - Pueden recibir parámetros pero **no pueden retornar ningún valor.**
  - Se **pueden sobrecargar**, es decir, se pueden definir varios constructores los cuales deben diferenciarse en la cantidad, tipo y orden de parámetros.
  - Se **nombran igual que la clase.**

```
class Esfera{
     //propiedades
     int radio;
     //constructores sobrecargados
     Esfera(){ this->radio = 0;}
     Esfera(int radio){ this->radio = radio;}
     //métodos de servicio, responsabilidades
     double area(){***}
     double volumen(){***}
}
```

Método: Constructor/Destructor

- Un **DESTRUCTOR** es un método especial que se invoca (llama) cuando la vida de un objeto termina.
- Propósito:
  - Liberar los recursos que el objeto pudiera haber adquirido durante su vida.

```
class Esfera{
     //propiedades
     int radio;
     //constructores sobrecargados
     Esfera(){ this->radio = 0;}
     Esfera(int radio){ this->radio = radio;}
     //Destructores vacío (no ejecuta tarea alguna)
     ~Esfera(){}
     //métodos de servicio, responsabilidades
     double area(){***}
     double volumen(){***}
}
```

Métodos: Set/Get

- Típicamente se utilizan para:
  - Obtener el estado de un atributo: recuperar el valor de un atributo (Método Get).
  - Asignar el estado a un objeto: modificar el valor de algún atributo (Método Set).

```
class Esfera{
     //propiedades
     int radio;
     //constructores sobrecargados
     Esfera(){ this->radio = 0;}
     Esfera(int radio){ this->radio = radio;}
     //Destructores vacío (no ejecuta tarea alguna)
     ~Esfera(){}
     //método set/get
     void setRadio(int radio){this->radio = radio}
     int getRadio(){return (this->radio);}
     //métodos de servicio, responsabilidades
     double area(){***}
     double volumen(){***}
}
```

Ciclo de Vida de los Objetos

- El proceso para crear objetos tiene dos etapas:
  - **Declaración de la variable:**
  - **Creación del objeto físico:**

```
class Esfera{
  //propiedades
  int radio;
  //constructores sobrecargados
  Esfera(){ this->radio = 0;}
  Esfera(int radio){ this->radio = radio;}
  //Destructores vacío
  ~Esfera(){}
  //métodos de servicio, responsabilidades
  double area(){***}
  double volumen(){***}
}
```

```
int main(){
  //Declaración de la variable
  Esfera *f;
  //Creación del objeto: constructor
  f = new Esfera(10);

  //Declaración y creación de un objeto
  Esfera *f1 = new Esfera();
  //Uso de los métodos de servicio
  f1->setRadio(10);
  cout<<"Esfera1: "<<f->area();
  cout<<"Esfera2: "<<f1->volumen();
  return(0);
}
```

Mensajes

- Un mensaje es una petición para solicitar una llamada a una función que pertenece a un objeto en particular.
- Todos los objetos de una determinada clase pueden recibir los mismos mensajes.
- Para enviar un mensaje al objeto, se referencia el nombre del objeto y agrega el nombre del método a ejecutarse mediante un punto.

Formas de envío de Mensaje en c++ :

Objeto.metodo(argumentos)

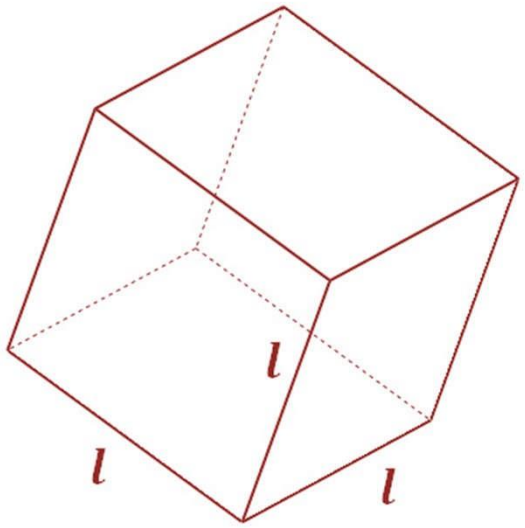
Objeto->metodo(argumentos)

```
f1->setRadio(10);
cout<<"Esfera1: "<<f->area();
cout<<"Esfera2: "<<f1->volumen();
```

Ejemplo

- Para la ABSTRACCION siguiente se pide:
  a) Graficar el Diagrama de Clases
  b) Implemente la clase

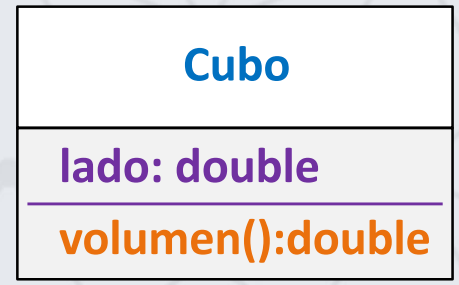
$$VOLUMEN = (l)^3$$



Ejemplo

cubo.h

```
#ifndef _CUBO_H_
#define _CUBO_H_
#include<cmath>
class Cubo{
  private:
    double lado;
  public:
    //Constructor y Destructor
     Cubo(){ this->lado = 0;}
    Cubo(double lado){ this->lado = lado;}
    ~Cubo(){}
    //Metodos Setter/Getter
    void setLado(double lad){
        lado = lad;
     }
     double getLado(){
          return(this->lado);
     }
    //Metodos de servicio
     double volumen(){
            return (pow(this->lado,3));
     }
};
#endif
```

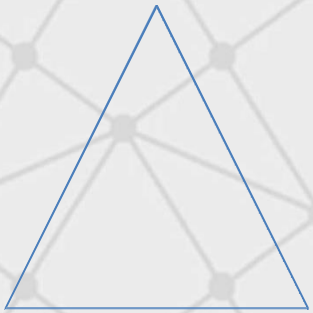


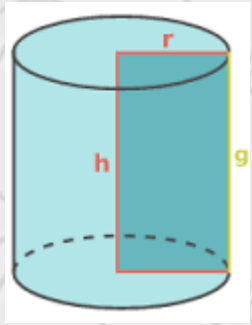
pruebaCubo.cpp

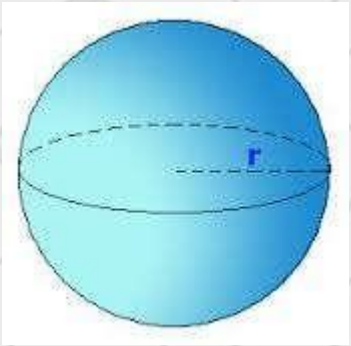
```
#include<iostream>
#include "cubo.h“
using namespace std;

int main(){
  Cubo* c1;
  c1 = new Cubo();
  c1->setLado(50);
  //
  Cubo* c2 =  new Cubo(20);
  //
  cout<<"Datos del Cubo 1"<<endl;
  cout<<"Lado: "<<c1->getLado()<<endl;
  cout<<"Volumen: "<<c1-volumen()<<endl;
    //
  cout<<"Datos del Cubo 2"<<endl;
  cout<<"Lado: " <<c2->getLado()<<endl;
  cout<<"Volumen: "<<c2->volumen()<<endl;
  return(0);
}
```

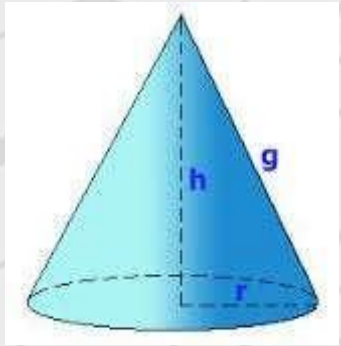
Ejercicios

- Tomando en cuenta el ejemplo desarrollado y dadas las ABSTRACCIONES siguientes se pide:
  a) Graficar el Diagrama de Clases
  b) Implemente la clase









Principios Fundamentales de la POO

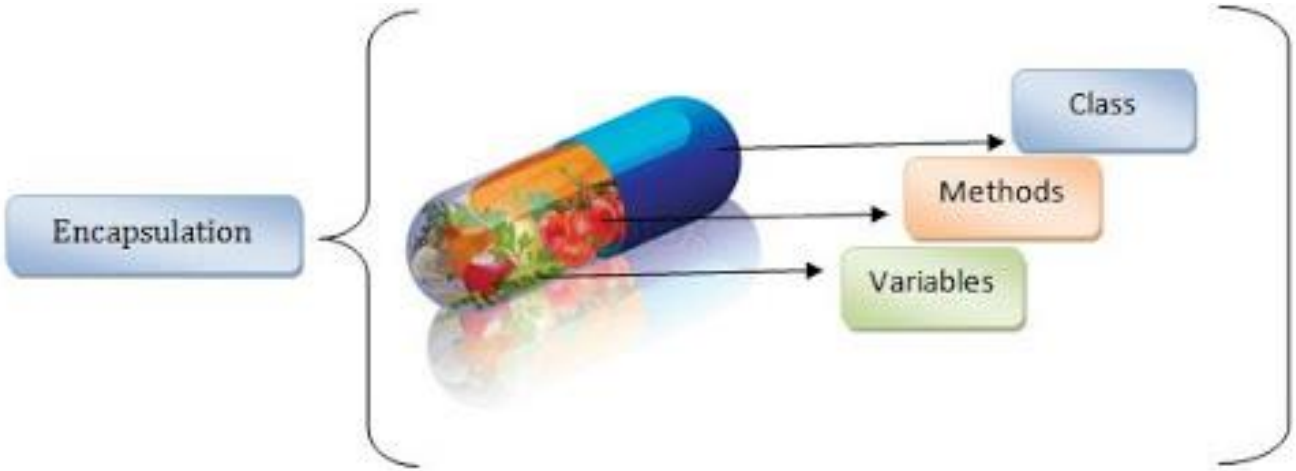
Consiste en la combinación de datos e instrucciones en un nuevo tipo de datos, denominado clase

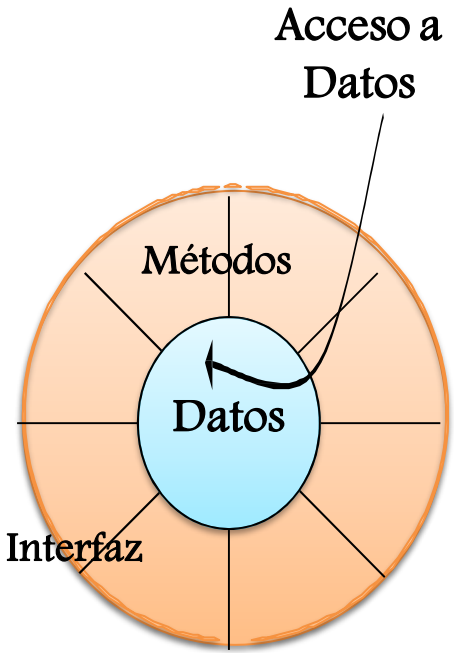
El acceso a los datos se realiza a través de los métodos de la Interfaz.

Reúne los elementos de un objeto que se consideren de una misma entidad, esto permite aumentar la cohesión de los componentes del sistema.

Principio #2: Encapsulamiento

- Consiste en la combinación de datos e instrucciones en un nuevo tipo de datos, denominado clase.
- El acceso a los datos se realiza a través de los métodos de la Interfaz.





Encapsulamiento

- Proceso por el que se ocultan:
  - Las estructuras de datos
  - Los detalles de la implementación

- Permite considerar a los objetos como "cajas negras", evitando que otros objetos accedan a detalles que NO LES INTERESA
- Una vez creada la clase, las funciones usuarias no requieren conocer los detalles de su implementación

Encapsulamiento

- Toda clase tiene un conjunto de ***atributos*** y ***métodos*** asociados a ella

- Todos ellos están ***encapsulados*** o contenidos dentro de la misma clase, de manera que son ***miembros*** de dicha clase

- Esos métodos y atributos pueden ser utilizados por ***otras*** clases ***sólo si*** la clase que los encapsula les brinda los ***permisos*** necesarios para ello

Encapsulamiento

- Formas de encapsular:
  - **Abierto** (public) : Los datos pueden ser accedidos sin restricción alguna.
  - **Protegido** (protected) : Los datos son accedidos bajo ciertas restricciones.
  - **Cerrado** (private) : Los datos no pueden ser accedidos

Principio #3: Herencia

Se encarga de organizar y facilitar el polimorfismo y encapsulamiento, de este modo permite crear objetos como tipos especiales de objetos predefinidos. Estos heredan las propiedades y comportamiento de su clase padre sin necesidad de volver a implementarlos

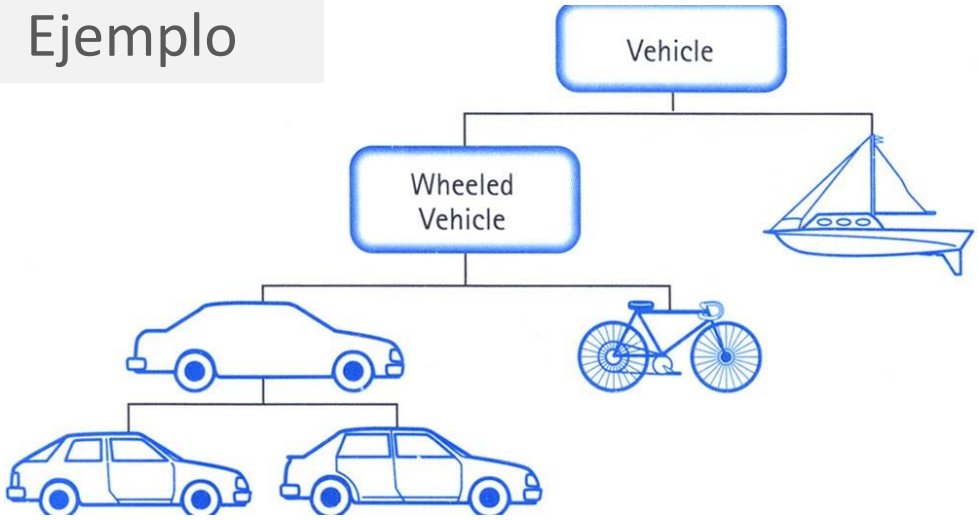
La Herencia define una relación entre clases donde una clase comparte la estructura y/o el comportamiento definido en una o más clases.

Principio #3: Herencia

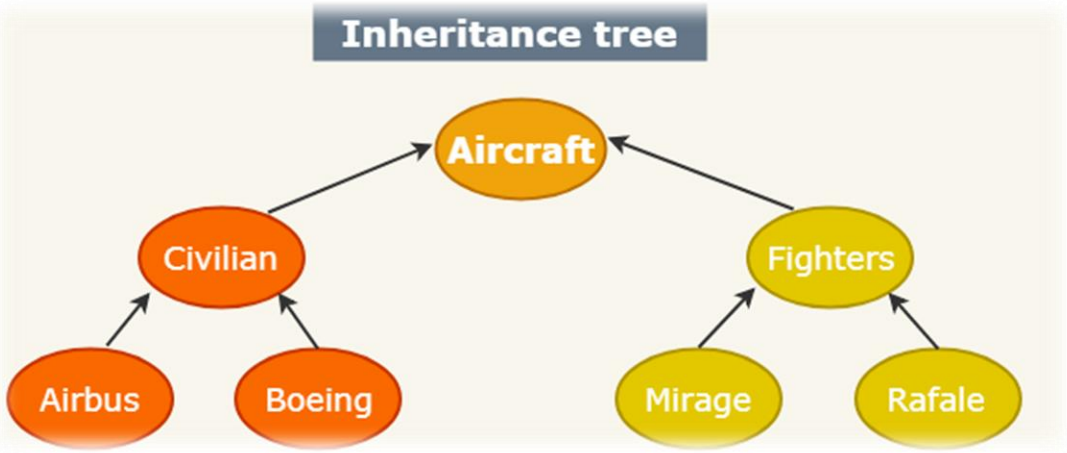
Ejemplo

Principio #3: Herencia

Ejemplo



Ejemplo



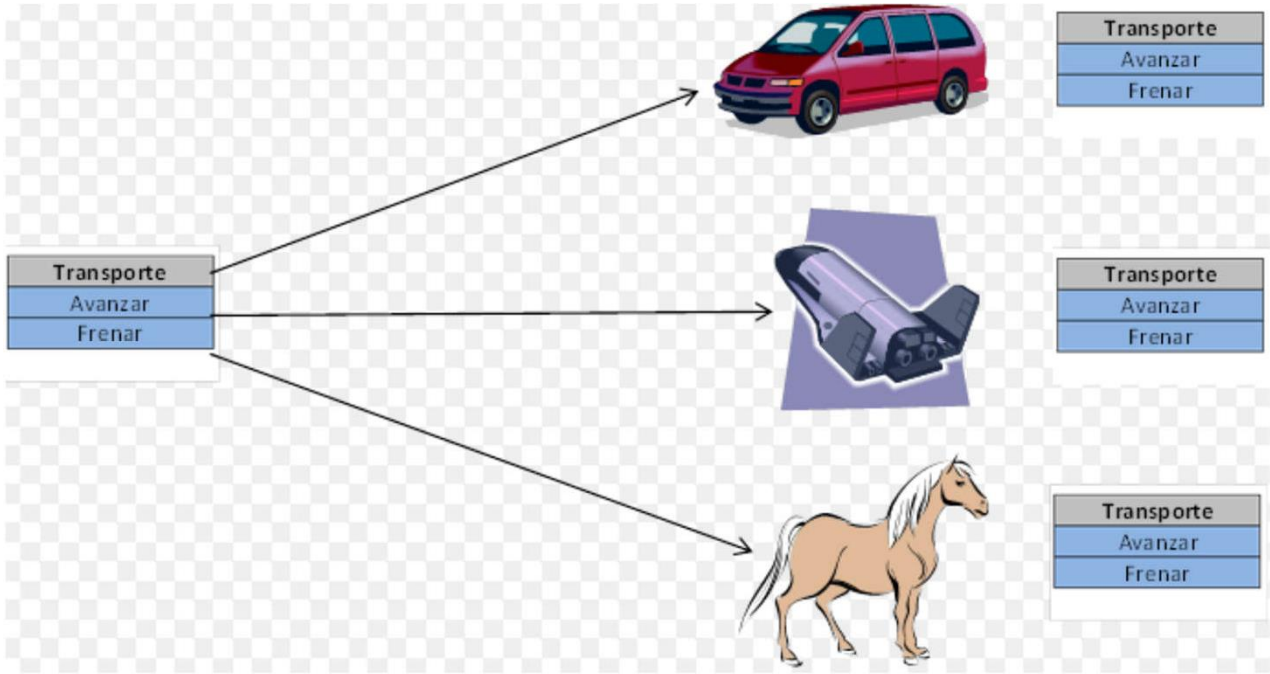
Principios Fundamentales de la POO

El polimorfismo permite que el mismo método ejecute diferentes comportamientos de dos formas: anulación de método y sobrecarga de método.

Son comportamientos diferentes de un objeto pero perteneciente a un mismo tipo de objetos, trabaja en conjunto con la herencia.

Principio #4: Polimorfismo

- Es la propiedad de los objetos de comportarse de forma específica ante un mensaje. Es decir, el mismo mensaje, a objetos diferentes desencadena comportamientos distintos.
- Permite que un objeto presente diferentes comportamientos en función del contexto en que se encuentre.



Ejercicios de Aplicación

Utilizando Programación Orientada a Objetos resovler lo siguiente:

1) Mover, en forma horizontal, un caracter en en la ventana de consola
2) Simular el juego de Sudoku (Ver un ejemplo de solución en el aula virtual)

Ejercicios de Aplicación

CLASE Hora

Esta clase debe permitir almacenar la hora, así como los métodos para manipularla. Tendrá las siguientes propiedades y métodos

Propiedades (privadas):

hora: de tipo entero (00 24)

minutos: de tipo entero (00 59)

segundos: de tipo entero (00 59)

Constructor

Constructor que, por defecto, inicialice las propiedades de la clase a 0

Constructor al que se le pasen como argumentos tres enteros y se los asigne a las propiedades de la clase. Si la cantidad recibida no satisface las restricciones de los valores impuestos a horas, minutos y segundos, el valor que se fija es 0

Ejercicios de Aplicación

Métodos de la clase (públicos):

setHora(): recibe como argumentos tres enteros y se los asigna a las propiedades de la clase. Utiliza el mismo nombre en las variables que reciben los argumentos y en las propiedades de la clase.

getHora(): devuelve la hora como arreglo [horas, minutos, segundos] o como un string de la forma "horas:minutos:segundos".

imprmirHora() que muestra en consola la hora en formato stringde la forma "horas:minutos:segundos".

Métodos set()y get() para todas las propiedades [Abstracción y encapsulamiento].