PROGRAMACIÓN ESTRUCTURADA

- **Descomposición del problema en procedimientos y funciones.**
- **El problema se resuelve con un conjunto de llamadas a estas funciones y procedimientos.**
- **Se usan Controles de Flujo.**
- **A partir de unos datos de entrada hay que conseguir unos datos de salida.**
- **Identifica las estructuras de datos que se necesitan.**

fppt.com

PROGRAMACIÓN ORIENTADA A OBJETOS

- **Se identifican los objetos abstractos que representan el problema.**
- **Se identifican las operaciones**
- **abstractas que hay que realizar con esos objetos.**
- **La solución al problema es una secuencia de llamadas a esos objetos.**
- **Código más robusto, fácil de mantener, seguro y reutilizable.**

fppt.com

PROGRAMACIÓN ORIENTADA A OBJETOS
DEFINICIONES

CLASE

- Es una representación real de un ***TDA*** *(tipo de dato abstracto).*
- Define ***atributos****(características)* y ***métodos****(comportamientos)* que implementa la estructura de datos.
- Define el modelo que van a seguir los Objetos que se instancien de ella.

Ejemplo:

CLASE: Vehículo
Atributos: Marca, Color, Velocidad, Tamaño, etc.
Funciones: Andar, Parar, Girar, etc.

CLASE: Persona
Atributos: Nombre, edad, estatura, estado civil, etc.
Funciones: Hablar, Caminar, Correr, Dormir, etc.

fppt.com

PROGRAMACIÓN ORIENTADA A OBJETOS
DEFINICIONES

ATRIBUTO$_{CLASE}$

- ***Características*** individuales que diferencian un objeto de otro y determinan su apariencia, estado u otras cualidades.
- (C++)Son las ***variables*** que se crean dentro de la clase, las cuales toman valores distintos para cada objeto.

Ejemplo:

CLASE: Vehículo
Atributos: Marca, Color, Velocidad, Tamaño.

C++ :
CLASE: Persona
Atributos: **char** Nombre, **int** edad, **float** estatura, **char** estado civil.

fppt.com

PROGRAMACIÓN ORIENTADA A OBJETOS
DEFINICIONES

MÉTODO$_{CLASE}$

- Definen el comportamiento de los ***objetos*** de una ***clase***, es un conjunto de instrucciones que realizan una determinada tarea y son similares a las funciones de los **lenguajes estructurados**.
- Además de dar la posibilidad de comportamiento a los objetos, son muy usados también para ir separando código que es muy extenso y en la definición de la clase se note mayor orden y quede más entendible. Y con esto ir dando pasos a la modularidad y reusabilidad.
- Son las operaciones (acciones o funciones) que se aplican sobre los objetos y que permiten crearlos, cambiar su estado o consultar el valor de sus atributos.

Ejemplo:

CLASE: Vehículo
METODO: andar( ), parar( ), girar( )

fppt.com

PROGRAMACIÓN ORIENTADA A OBJETOS
DEFINICIONES

OBJETO

- Un ***objeto*** es una instancia de una ***clase***.
- Puede ser identificado en forma única por su nombre, el cuál es representado por los valores de sus atributos en un momento en particular.
- Al ser una ***instancia*** de una clase, es el que decide cómo se va a comportar de acuerdo a lo que le haya definido la **clase**.

Ejemplo:
CLASE: Persona
Atributos: Nombre, edad, estatura, estado civil, etc.
Funciones: Hablar, Caminar, Correr, Dormir, etc.

OBJETO: Luis, 25 años, 1.65 mts, soltero.
Hablar(1 hora), Camina r(3 km), Correr(1 milla), Dormir(8 h.)

DIAGRAMA DE CLASES

DIAGRAMA DE CLASES

- El propósito de este diagrama es el de representar los objetos fundamentales del sistema, es decir los que percibe el usuario y con los que espera tratar para completar su tarea en vez de objetos del sistema o de un modelo de programación.
- La clase define el ámbito de definición de un conjunto de objetos.
- Cada objeto pertenece a una clase.
- Los objetos se crean por instanciación de las clase.

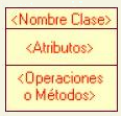
REPRESENTACION DE CLASES
ELEMENTOS

Una clase es la unidad básica que encapsula toda la información de un Objeto (un objeto es una instancia de una clase). A través de ella podemos modelar el entorno en estudio (una Casa, un Auto, una Cuenta Corriente, etc.).

El Lenguaje Unificado de Modelado o UML, es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema; se pueden representar: clase, objetos, componentes, paquetes; los cuales en el mundo real son abstractos, el UML consiste en un complemento de la programación orientada a objetos.

REPRESENTACION DE CLASES
ELEMENTOS

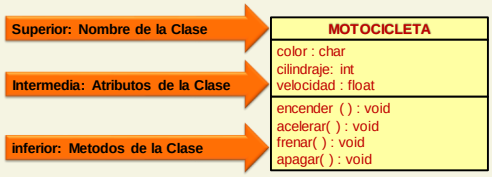
En UML, una clase es representada por un rectángulo que posee tres divisiones:



- ***Superior***: Contiene el nombre de la Clase.
- ***Intermedio***: Contiene los atributos (o variables de instancia) que caracterizan a la Clase (pueden ser **private**, **protected** o **public**).
- ***Inferior***: Contiene los métodos u operaciones, los cuales son la forma como interactúa el objeto con su entorno (dependiendo de la visibilidad: **private**, **protected** o **public**).

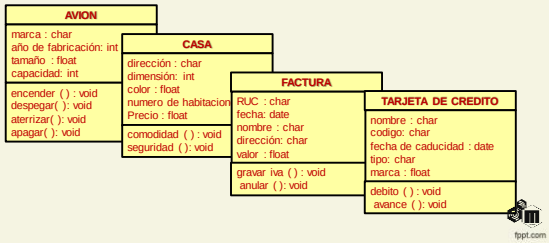
REPRESENTACION DE CLASES
ELEMENTOS

A continuación se muestra un ejemplo de la representación UML de una clase "MOTOCICLETA"; nótese sus tres partes y la forma de cómo escribir sus elementos.



REPRESENTACION DE CLASES
EJERCICIOS

Realicemos la representación de las siguientes clases: **Avión**, **Casa** , **Factura**, **Tarjeta de Crédito**.



fppt.com

REPRESENTACION DE CLASES
MODOS DE ACCESO

Los modos de acceso definen el grado de comunicación y visibilidad de ellos con el entorno así también como la forma de interactuar con el entorno.

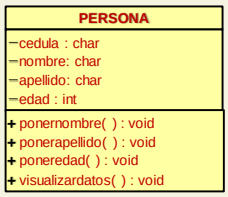
Los modos de acceso son:

+ **Público**: **(Atributos o Métodos) que son accesibles fuera de la clase. Pueden ser llamados por cualquier clase, aun si no está relacionada con ella.**
- **Privado**: **(Atributos o Métodos) que sólo son accesibles dentro de la implementación de la clase.**
Protegido: **(Atributos o Métodos) que son accesibles para la propia clase y sus clases hijas (subclases).**

fppt.com

REPRESENTACION DE CLASES
MODOS DE ACCESO

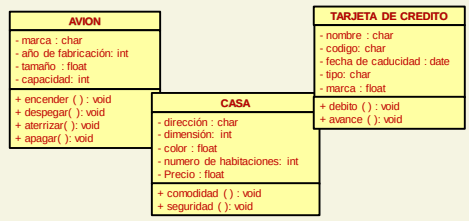
Realicemos la representación de la clase **Persona**, con sus métodos de acceso:



+ **Público (PUBLIC)**
- **Privado (PRIVATE)**
Protegido (PROTECTED)

fppt.com

REPRESENTACION DE CLASES
MODOS DE ACCESO



fppt.com

fppt.com

REPRESENTACION DE CLASES
LENGUAJE C++

 # librerías

```
// Definicion de la Clase
class NOMBRE_CLASE
{
  private:
    // Atributos
        ......
        ......
  public:
    // Funciones o Metodos
        ......
        ......
};

// Implementacion de Funciones o Metodos (Clase)
Tipodato NOMBRE_CLASE::método ( ____)
{
...........
}

// Programa Principal
int main()
{
   NOMBRE_CLASE obj;
.....
getch();
return 0;
}
```

fppt.com

REPRESENTACION DE CLASES LENGUAJE C++

PERSONA

- nombre: char
- apellido: char
- cedula : char
- edad : int

+ llenarNombre(nombre) : void
+ llenarApellido(apellido) : void
+ llenarCedula(cedula) : void
+ llenarEdad(edad): void
+ visualizardatos ( ) : void

```
#include <iostream.h>
#include <conio.h>

class Persona
{
  private:
    // Atributos
    char *nombre, *apellido, *cedula;
    int edad;

  public:

    // Funciones o Metodos
      void llenarNombre( char *n);
      void llenarApellido(char *a);
      void llenarCedula(char *c);
      void llenarEdad(int e);
      void visualizardatos();
};
```

```
//Llenar datos
void Persona::llenarNombre(char *n)
{
   nombre=n;
}

void Persona::llenarApellido(char *a)
{
   apellido=a;
}

void Persona::llenarCedula(char *c)
{
   cedula=c;
}

void Persona::llenarEdad(int e)
{
   edad=e;
}

// implementacion muestra datos
void Persona::visualizardatos()
{
 cout<<nombre<<endl;
 cout<<apellido<<endl;
 cout<<cedula<<endl;
 cout<<edad<<endl;
}
```

```
//Programa Principal

int main()
{
   Persona P

   P.llenarNombre("Alberto");
   P.llenarApellido("Arellano");
   P.llenarCedula("123456789-0");
   P.llenarEdad(35);

   P.visualizardatos();

  getch();
  return 0;
}
```

fppt.com

REPRESENTACION DE CLASES C++ EJERCICIOS

Realizar un programa con la clase VEHICULO, crear sus atributos y métodos necesarios para acceder a dicha clase; crear un objeto para la clase e inicializar sus valores

Realizar un programa con la clase CASA, crear sus atributos y métodos necesarios para acceder a dicha clase; crear un objeto para la clase e inicializar sus valores

fppt.com

INICIALIZACIÓN Y RECUPERACION DE DATOS EN UNA CLASE

fppt.com

REPRESENTACION DE CLASES LENGUAJE C++ set get

PUNTO2D

- x : int
- y : int

+ setX(x): void
+ setY(y) : void
+ getX() : int
+ getY(): int
+ vizualizarpunto() : void

```
#include <iostream.h>
#include <conio.h>

class PUNTO2D
{
  private:
    // Atributos
    int x, y;

  public:

    // Funciones o Metodos
      void setX( int xx);
      void setY(int yy);
      int getX();
      int getY();
      void visualizarpunto();
};
```

```
//Llenar datos
void PUNTO2D::setX(int xx)
{
   x=xx;
}

void PUNTO2D::setY(int yy)
{
   y=yy;
}

int PUNTO2D::getX()
{
   return x;
}

int PUNTO2D::getY()
{
   return y;
}

// implementacion muestra datos
void PUNTO2D::visualizarpunto()
{
cout<<"( ";
cout<<x;
cout<<" , ";
 cout<<y;
cout<<")"<<endl;
}
```

```
//Programa Principal

int main()
{
   PUNTO2D obj;

   obj.setX(2);
   obj.setY(3);

   cout<<obj.getX()<<endl;
   cout<<obj.getY()<<endl;

   obj.visualizarpunto();

  getch();
  return 0;
}
```

fppt.com

El tiempo en cualquier reloj se mide en horas, minutos y segundos; Implementar un programa que me permita inicializar y recuperar un tiempo especifico. (usar clases).
Mostrar un tiempo *ingresado* (objeto) en los siguientes formatos:

1

Básico: SSMMHH
Extendido: SS-MM-HH
Internacional: HH-MM-SS
Análoga: HH,MM,SS
Digital: HH:MM:SS

2

La fecha en cualquier calendario esta descrita por días, meses y años. Implementar un programa que me permita inicializar y recuperar una fecha especifica. (usar clases).
Mostrar un fecha *ingresada* (objeto) en los siguientes formatos:

Básico: AAAAMMDD
Extendido: AAAA-MM-DD
Internacional: AAAA/MM/DD
Calendario: DD/MM/AAAA
Digital: DD-MM-AAAA

EJERCICIOS

fppt.com

fppt.com

RELACIONES ENTRE CLASES
DEFINICIONES

Las clases no se construyen para que trabajen de manera aislada, la idea es que ellas se puedan relacionar entre sí, de manera que puedan compartir atributos y métodos sin necesidad de rescribirlos.

La posibilidad de establecer jerarquías entre las clases es una característica que diferencia esencialmente la programación orientada a objetos de la programación tradicional, ello debido fundamentalmente a que permite extender y reutilizar el código existente sin tener que rescribirlo cada vez que se necesite.

Los cuatro tipos de ***relaciones entre clases*** existentes son:

- ✓ Herencia (*Generalización / Especialización o Es-un*)
- ✓ Agregación (*Todo / Parte o Forma-parte-de*)
- ✓ Composición (*Es parte elemental de*)
- ✓ Asociación (*entre otras, la relación Usa-a*)

fppt.com

RELACIONES ENTRE CLASES
HERENCIA
(Generalización / Especialización, Es un)

Superclase. Es la ***clase principal*** o llamada ***clase padre***, y es la que a partir de ella se pueden generar herencias

Subclase. Es la clase hijas, capaz de heredar atributos y métodos de una clase padre o superclase.

La ***Herencia*** es un tipo de jerarquía de clases, en la que cada ***subclase*** contiene los atributos y métodos de una (herencia simple) o más ***superclases*** (herencia múltiple).

Mediante la ***herencia***, las instancias de una clase hija (o subclase) pueden acceder tanto a los atributos como a los métodos públicos y protegidos de la clase padre (o superclase). Cada subclase o clase hija en la jerarquía es siempre una extensión (esto es, conjunto estrictamente más grande) de la(s) superclase(s) o clase(s) padre(s) y además incorporar atributos y métodos propios, que a su vez serán heredados por sus hijas.

fppt.com

RELACIONES ENTRE CLASES
HERENCIA
(Generalización / Especialización, Es un)

Superclase. Es la ***clase principal*** o llamada ***clase padre***, y es la que a partir de ella se pueden generar herencias

Subclase. Es la clase hijas, capaz de heredar atributos y métodos de una clase padre o superclase.

La ***Herencia*** es un tipo de jerarquía de clases, en la que cada ***subclase*** contiene los atributos y métodos de una (herencia simple) o más ***superclases*** (herencia múltiple).

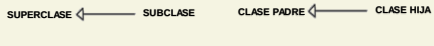
Mediante la ***herencia***, las instancias de una clase hija (o subclase) pueden acceder tanto a los atributos como a los métodos públicos y protegidos de la clase padre (o superclase). Cada subclase o clase hija en la jerarquía es siempre una extensión (esto es, conjunto estrictamente más grande) de la(s) superclase(s) o clase(s) padre(s) y además incorporar atributos y métodos propios, que a su vez serán heredados por sus hijas.

fppt.com

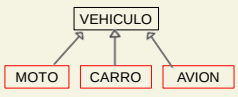
RELACIONES ENTRE CLASES
HERENCIA
(Generalización / Especialización, Es un)

REPRESENTACION UML DE HERENCIA DE CLASES

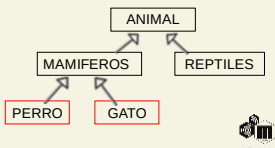
El diagrama de clases para herencias en UML, se representa mediante una relación de *generalización/especificación*, que se denota de la siguiente forma:



HERENCIA SIMPLE



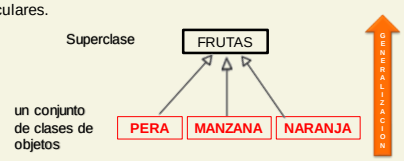
HERENCIA MULTIPLE



fppt.com

RELACIONES ENTRE CLASES
HERENCIA
(Generalización / Especialización, Es un)

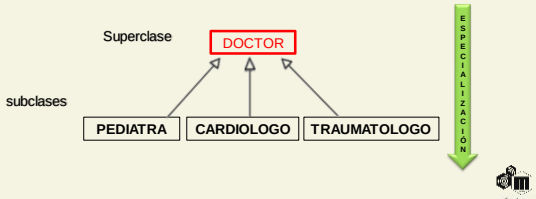
Generalización. es el mecanismo de abstracción mediante el cual ***un conjunto de clases de objetos*** son agrupados en ***una clase de nivel superior*** (Superclase), donde las semejanzas de las clases constituyentes (Subclases) son enfatizadas, y las diferencias entre ellas son ignoradas. En consecuencia, a través de la generalización, la superclase almacena datos generales de las subclases, y las subclases almacenan sólo datos particulares.



fppt.com

RELACIONES ENTRE CLASES
HERENCIA
(Generalización / Especialización, Es un)

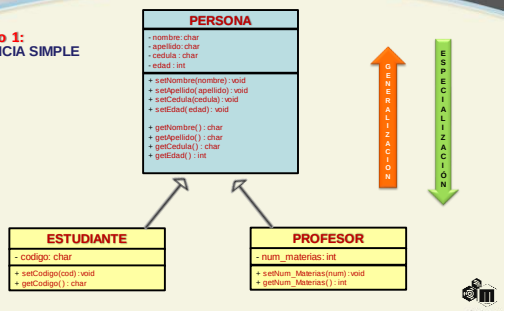
Especialización. es lo contrario de la generalización. *A partir de una Superclase*(padre) se pueden subdividir en *subclases*(hijas); estas subclase poseen características propias de ellas; sin embargo también poseen algunas características comunes las cuales se las refleja en la superclase.



fppt.com

RELACIONES ENTRE CLASES
HERENCIA
(Generalización / Especialización, Es un)

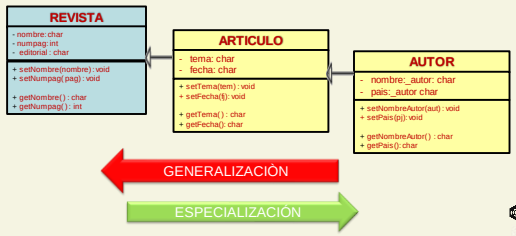
Ejemplo 1:
HERENCIA SIMPLE



fppt.com

RELACIONES ENTRE CLASES
HERENCIA
(Generalización / Especialización, Es un)

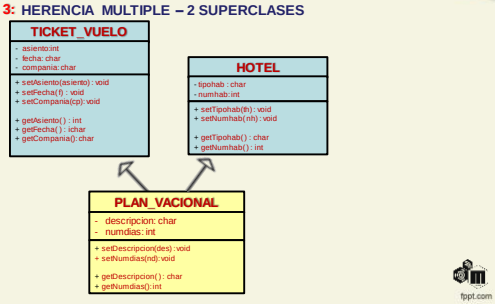
Ejemplo 2:
HERENCIA MULTIPLE (NIVELES)



fppt.com

RELACIONES ENTRE CLASES
HERENCIA
(Generalización / Especialización, Es un)

Ejemplo 3: HERENCIA MULTIPLE – 2 SUPERCLASES



fppt.com

RELACIONES ENTRE CLASES
HERENCIA
(Generalización / Especialización, Es un)

PRACTICA

Realizar la Implementación el *Ejemplo 1* : Herencia Simple, utilice programación *inline* para optimizar líneas de codigo.

Realizar la Implementación el *Ejemplo 2* Herencia Múltiple (2 niveles). utilice programación *inline* para optimizar líneas de codigo.

Realizar la Implementación el *Ejemplo 3* Herencia Múltiple(2 superclases). utilice programación *inline* para optimizar líneas de codigo.

fppt.com

PROGRAMACIÓN C++
TÓPICO
Programación inline

La programación *inline*, es generalmente utilizado para la optimización de número de líneas de código

Permite una ejecución mas rápida del programa.

Permite la implementación de funciones y procedimientos al mismo momento de llamarlos.

Progam. inline	case default: {cout<<"OPCIO NO EXISTE";}
Procd.. inline	error() {cout<<"DATOS MAL INGRESADOS";}
Func. inline	suma(a,b) {return a+b;}

Luego de los corchetes { } no existe el punto y coma.
Antes del corchete final } siempre existe un punto y coma.

fppt.com

Realizar la Implementación el Ejemplo 1 : Herencia Simple,
utilice programación *inline* para optimizar líneas de codigo.

```
# include <iostream.h>
# include <conio.h>
# include <stdlib.h>
# include <stdio.h>

// Super Clase
class Persona
{
   private:
    // Atributos
    char *nombre, *apellido, *cedula;
    int edad;

   public:

    // Funciones o Metodos inlie
    void setNombre(char  *n){nombre=n;}
    void setApellido(char  *a){apellido=a;}
    void setCedula(char  *c){cedula=c;}
    void setEdad(int  e){edad=e;}

    char *getNombre(){return  nombre;}
    char *getApellido(){return  apellido;}
    char *getCedula(){return  cedula;}
    int  getEdad(){return  edad;}
};
```

```
//Sub clase estudiante
class Estudiante:  public Persona
{
   private:
         char *codigo;

   public:
         void setCodigo(char  *cd){codigo=cd;}
         char *getCodigo(){return  codigo;}
};

//Sub clase Docente
class Docente:  public Persona
{
private:
        int num_materias;

public:
  void setNum_materias(int  nm){num_materias=nm;}
    int getNum_materias(){return  num_materias;}
};
```

```
int main()
{
Estudiante estudiante;
Docente docente;

cout<<"OBJETO  ESTUDIANTE"<<endl;

estudiante.setNombre("Juan");
estudiante.setApellido("Lopez");
estudiante.setCedula("123456789-0");
estudiante.setEdad(25);
estudiante.setCodigo("2050");

cout<<estudiante.getNombre()<<endl;
cout<<estudiante.getApellido()<<endl;
cout<<estudiante.getCedula()<<endl;
cout<<estudiante.getEdad()<<endl;
cout<<estudiante.getCodigo()<<endl;

cout<<endl;

cout<<"OBJETO  DOCENTE"<<endl;
docente.setNombre("Edwin");
docente.setApellido("Mejia");
docente.setCedula("123456789-0");
docente.setEdad(39);
docente.setNum_materias(40);

cout<<docente.getNombre()<<endl;
cout<<docente.getApellido()<<endl;
cout<<docente.getCedula()<<endl;
cout<<docente.getEdad()<<endl;
cout<<docente.getNum_materias()<<endl;

getch();
return 0;
}
```

fppt.com

Realizar la Implementación el Ejemplo 2 : Herencia Múltiple (2 NIVELES), utilice programación *inline* para optimizar líneas de codigo.

```
# include <iostream.h>
# include <conio.h>
# include <stdlib.h>
# include <stdio.h>

// Super Clase
class Revista
{
   private:
     // Atributos
     char *nombrerevista;
     int numpag;
     char *editorial;

   public:
     // Funciones o Metodos inlie
        void setNombrerevista(char
*nr){nombrerevista=nr;}
        void setNumpag(int np){numpag=np;}
        void setEditorial(char *ed){editorial=ed;}

        char *getNombrerevista(){return
nombrerevista;}
        int getNumpag(){return numpag;}
        char *getEditorial(){return editorial;}
};
```

```
//Subclase Articulo
class Articulo: public Revista
{
 private:
        char *tema;
        char *fecha;

 public:
        void setTema(char *t){tema=t;}
        void setFecha(char *f){fecha=f;}

        char *getTema(){return tema;}
        char *getFecha(){return fecha;}
};


//Subclase tercer nivel Autor
class Autor: public Articulo
{
 private:
        char *nombre_autor;
        char *pais;

 public:
        void setNombre_autor(char *na){nombre_autor=na;}
        void setPais(char *ps){pais=ps;}

        char *getNombre_autor(){return nombre_autor;}
        char *getPais(){return pais;}
};
```

```
//programa principal
int main()
{
 Autor autor;

 cout<<"CREAR OBJETOS COMPLETO "<<endl<<endl;

 autor.setNombrerevista("DISEÑO GRAFICO Y
TECNOLOGIA");
 autor.setNumpag(98);
 autor.setEditorial("ELSEVIER");

 autor.setTema("DISÑEO EN VIDRIO");
 autor.setFecha("02/06/2012");

 autor.setNombre_autor("LUIS ALMEIDA");
 autor.setPais("ECUADOR");

 cout<<autor.getNombrerevista()<<endl;
 cout<<autor.getNumpag()<<endl;
 cout<<autor.getEditorial()<<endl;
 cout<<autor.getTema()<<endl;
 cout<<autor.getFecha()<<endl;
 cout<<autor.getNombre_autor()<<endl;
 cout<<autor.getPais()<<endl;

getch();
return 0;
}
```

fppt.com

Realizar la Implementación el Ejemplo 3 : Herencia Múltiple (2 superclases), utilice programación *inline* para optimizar líneas de codigo.

```
#include <iostream.h>
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>

//1ª ssuperclase
class TICKET_VUELO
{
  private:
     int asiento;
     char *fecha;
     char *compania;


  public:

void setAsiento(int as){asiento=as;}
void setFecha(char *f){fecha=f;}
void setCompania(char *cp){compania=cp;}

int getAsiento(){return asiento;}
char *getFecha(){return fecha;}
char *getCompania(){return compania;}


};
```

```
//2ª ssuperclase
class HOTEL
{
    private:
        char *tipohab;
        int numhab;
    public:
        void setTipohab(char *th){tipohab=th;}
        void setNumhab(int nh){numhab=nh;}

        char *getTipohab(){return tipohab;}
        int getNumhab(){return numhab;}
};


// subclase
class PLAN_VACACIONAL: public TICKET_VUELO, public HOTEL
{
   private:

        char *descripcion;
        int numdias;
   public:
        void setDescripcion(char *ds){descripcion=ds;}
        void setNumdias(int nd){numdias=nd;}

        char *getDescripcion(){return descripcion;}
        int getNumdias(){return numdias;}
};
```

```
int main()
{
  PLAN_VACACIONAL  plan;

  plan.set~~;
  cout<<plan.get~~~~;

getch();
return 0;
}
```

fppt.com

REPRESENTACION DE CLASES
CONSTRUCTOR Y DESTRUCTOR

- Una ***clase*** define un ***modelo*** de creación, define una ***estructura*** de creación, define un ***esquema*** de creación, define una ***forma*** de creación, de algo mas particular llamado **objeto**.
- Un ***objeto*** es una instancia de una ***clase***, estos ***objetos*** se crean a partir de la definición de una ***clase***; por tanto todo o algún ***objeto*** siempre pertenecerá a una ***clase ;*** el ***objeto*** se creará según las reglas de la ***clase***

Para que una clase admita la creación de uno o varios objetos, se necesita del CONSTRUCTOR y DESTRUCTOR, estas son funciones necesarias que permitirán controlar la existencia de objetos de una clase.

Constructor.- Función miembro de la clase que se ejecuta cuando se crea un objeto.

Destructor.- Función miembro de la clase que se ejecuta al final de la vida de cada objeto.

fppt.com

REPRESENTACION DE CLASES
CONSTRUCTOR Y DESTRUCTOR

El ***constructor*** y ***el destructor*** son funciones miembro, por lo que tienen acceso directo a las variables miembro privadas de la clase.

El nombre del ***constructor*** y del ***destructor*** son siempre el nombre de la ***clase.***

Los ***constructores*** y ***destructores*** se llaman automáticamente siempre que se crea un objeto de una clase.

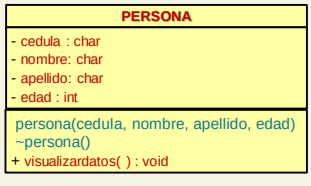
Un ***constructor*** invoca a todos los ***atributos*** existentes en una clase.
Persona(cedula, nombre, apellido, edad)

Un ***destructor*** es vacío y su labor es ***liberar*** un ***objeto*** después de haber sido manipulado. ***~Persona( )***

fppt.com

REPRESENTACION DE CLASES
CONSTRUCTOR Y DESTRUCTOR

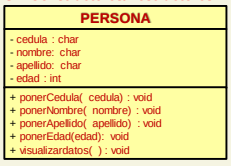
Realicemos la representación de la clase **Persona**, con sus métodos de acceso y con su respectivo constructor y destructor.



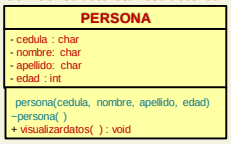
fppt.com

REPRESENTACION DE CLASES
CONSTRUCTOR Y DESTRUCTOR

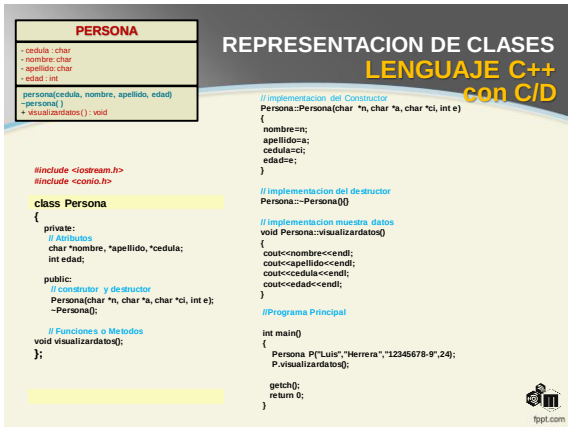
Sin Constructores/Destructores



Con Constructores/Destructores



fppt.com


PERSONA
- cedula : char
- nombre: char
- apellido: char
- edad : int
persona(cedula, nombre, apellido, edad)
~persona( )
+ visualizardatos( ) : void
REPRESENTACION DE CLASES
LENGUAJE C++
con C/D
#include <iostream.h>
#include <conio.h>
class Persona
{
private:
// Atributos
char *nombre, *apellido, *cedula;
int edad;
public:
// construtor y destructor
Persona(char *n, char *a, char *ci, int e);
~Persona();
// Funciones o Metodos
void visualizardatos();
};
// implementacion del Constructor
Persona::Persona(char *n, char *a, char *ci, int e)
{
nombre=n;
apellido=a;
cedula=ci;
edad=e;
}
// implementacion del destructor
Persona::~Persona()
// implementacion muestra datos
void Persona::visualizardatos()
{
cout<<nombre<<endl;
cout<<apellido<<endl;
cout<<cedula<<endl;
cout<<edad<<endl;
}
//Programa Principal
int main()
{
Persona P("Luis","Herrera","12345678-9",24);
P.visualizardatos();
getch();
return 0;
}
fppt.com


Ingeniería en
Tegnologías de
la Información
GRACIAS
Ing. Diego Reina
fppt.com