Programación Orientada a Objetos – Manual de Ejercicios en Clase

Luis Zelaya

Segunda Edición

10/05/2009

TABLA DE CONTENIDO

PRIMER PARCIAL

Repaso de Programación Estructurada

Conceptos Básicos

Programa

Es un conjunto de instrucciones o estipulaciones (también llamadas código) ejecutadas por la CPU de la computadora.
Estas instrucciones pueden ser escritas en muchos lenguajes diferentes. Luego deben ser convertidas a un lenguaje comprensible para la máquina.

Algoritmo

Series de pasos para resolver problemas. Los pasos de una solución (instrucciones) permanecen iguales, ya sea que esté resolviéndolos por computadora o a mano.

Funciones

Son la expresión de los algoritmos en algún lenguaje de programación específico.
Se vuelven a utilizar las funciones cuando se necesitan. No es necesario reescribir cada vez las líneas de código representadas por la función.

Estructuras de Programación

- **Estructura de Secuencia**: La computadora ejecuta líneas de código en el orden en que están escritas
- **Estructuras de Selección**: Se construyen en base a una declaración condicional. Si la condición es verdadera, ciertas líneas de código son ejecutadas. Si es falsa, esas líneas no se ejecutan.
- **Estructuras de Repetición**: Se construyen en base a instrucciones condicionales. Si la condición es verdadera un bloque de uno o más comandos se repite hasta que la condición es falsa.

Pasos para la automatización de un problema

1. Planteamiento del Problema
2. Análisis del Problema
3. Algoritmo de Resolución
4. Programación
5. Ejecución por Computadora

Primer Ejemplo de automatización de un problema mediante un programa (Estructura de Secuencia y de Selección)

Planteamiento del Problema

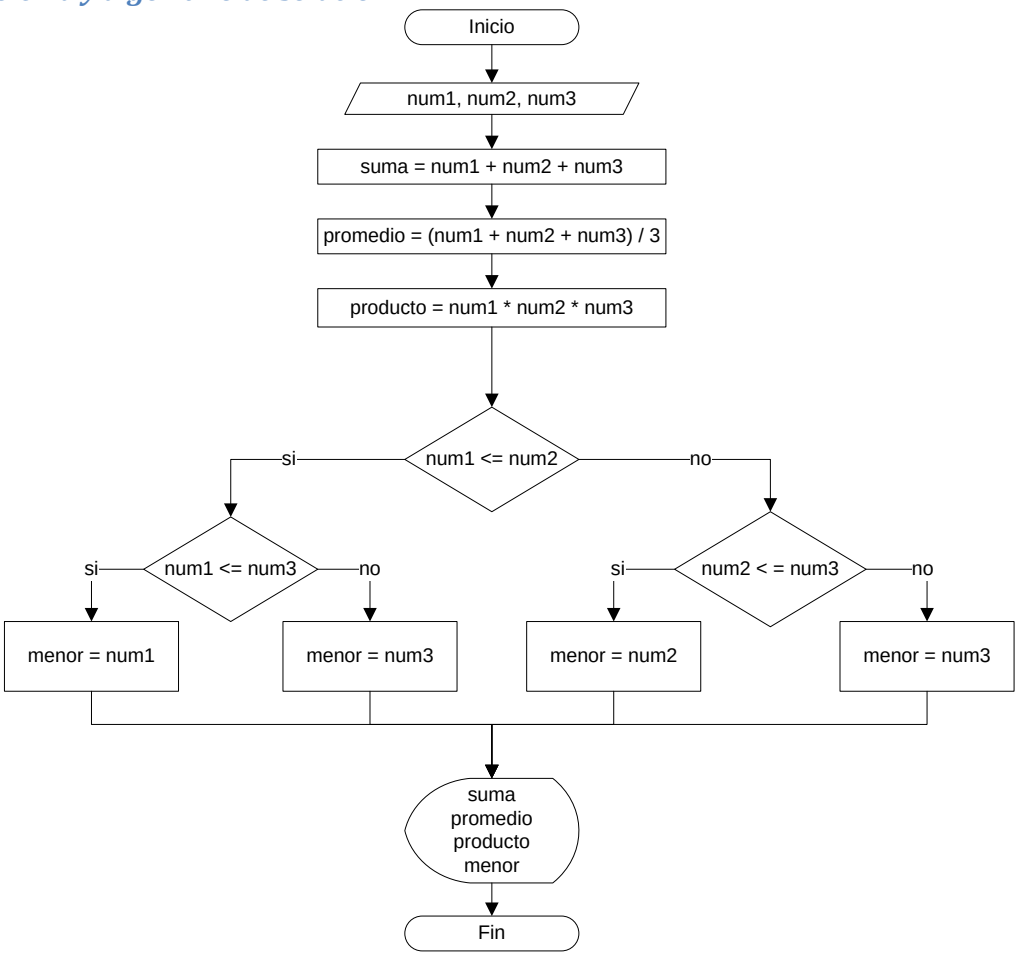
Dados tres números calcule:

- La suma de los tres
- El promedio de los tres
- El producto de los tres

Además determine:

- El menor de los tres

Análisis del Problema y algoritmo de solución



Código fuente del programa

PrimerEjemploProgramacion.cpp

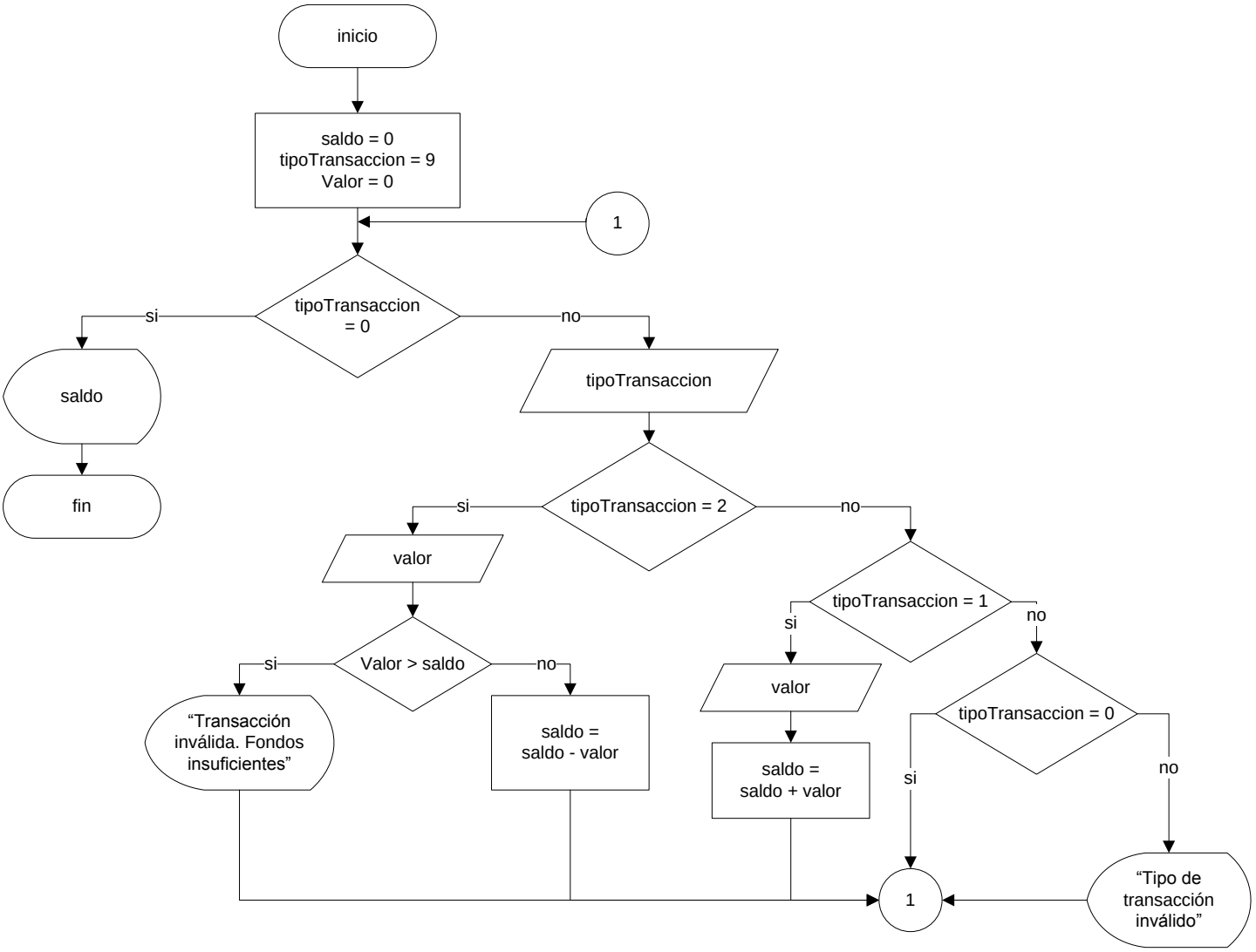
```
#include <iostream>
using std::cout;  // el programa utiliza cout
using std::cin;   // el programa utiliza cin
using std::endl;  // el programa utiliza endl

// la función main comienza la ejecución del programa
int main()
{
   int num1, num2, num3;  // variables de entrada: números que introduce el usuario
   int suma, promedio, producto, menor; // variables de proceso y salida
   cout << "Digite tres números enteros diferentes: \n";
   cin >> num1 >> num2 >> num3;   // lee tres enteros
   // Suma
   suma = num1 + num2 + num3;
   // Promedio
   promedio = (num1 + num2 + num3) / 3;
   // Producto
   producto = num1 * num2 * num3;
   // Menor
         if ((num1 <= num2) && (num1 <= num3))
          menor = num1;
         else
            if ((num2 <= num1) && (num2 <= num3))
                     menor = num2;
            else
                     menor = num3;
      cout << "La suma es igual a: " << suma << endl;
         cout << "El promedio es igual a: " << promedio << endl;
         cout << "El producto es igual a: " << producto << endl;
         cout << "El menor número es: " << menor << endl;
       system("pause"); // indica que el programa terminó satisfactoriamente
} // fin de la función main
```

Segundo Ejemplo de automatización de un problema mediante un programa (Estructura de Repetición y de Selección)

Planteamiento del Problema

Se necesita un programa que permita manejar transacciones de una cuenta.
El saldo inicial de la cuenta debe ser de Lps. 0.00
El programa debe solicitar al usuario que indique si desea realizar un depósito o un retiro.
Si el usuario elige hacer un retiro, se solicita un valor y debe verificarse que haya saldo suficiente para retirar. De no ser así se envía un mensaje al usuario notificando esa situación. Si hay saldo suficiente, se resta el valor ingresado al saldo.
Si el usuario elige hacer un depósito se solicita un valor y ese valor se suma al saldo.
Al final de cada transacción se pregunta al usuario si desea realizar otra transacción. Si contesta afirmativamente, se repiten las acciones anteriores. Si no, se termina el programa, mostrando el saldo final de la cuenta.

Análisis del problema y algoritmo de solución



Código fuente del programa

```
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

int main()
{
	int saldo = 0;
	int tt = 9;
	int valor = 0;

	while ( tt != 0)
	{
		cout << "Tipo de Transacción (1=Deposito, 2=Retiro, 0=Salir del Programa): ";
		cin >> tt;


		if (tt == 2)
		{
			cout << "\n Valor de Transacción: ";
			cin >> valor;

			if (valor > saldo)
				cout << "Transacción inválida. Fondos insuficientes" << endl;
			else
				saldo = saldo - valor;
		}
		else
			if (tt == 1)
			{
				cout << "\n Valor de Transacción: ";
				cin >> valor;
				saldo = saldo + valor;
			}

			else
				if (tt != 0)
					cout << "Tipo de Transacción inválido" << endl;
	}

	cout << "Saldo Final: " << saldo << endl;

	system("pause");
}
```

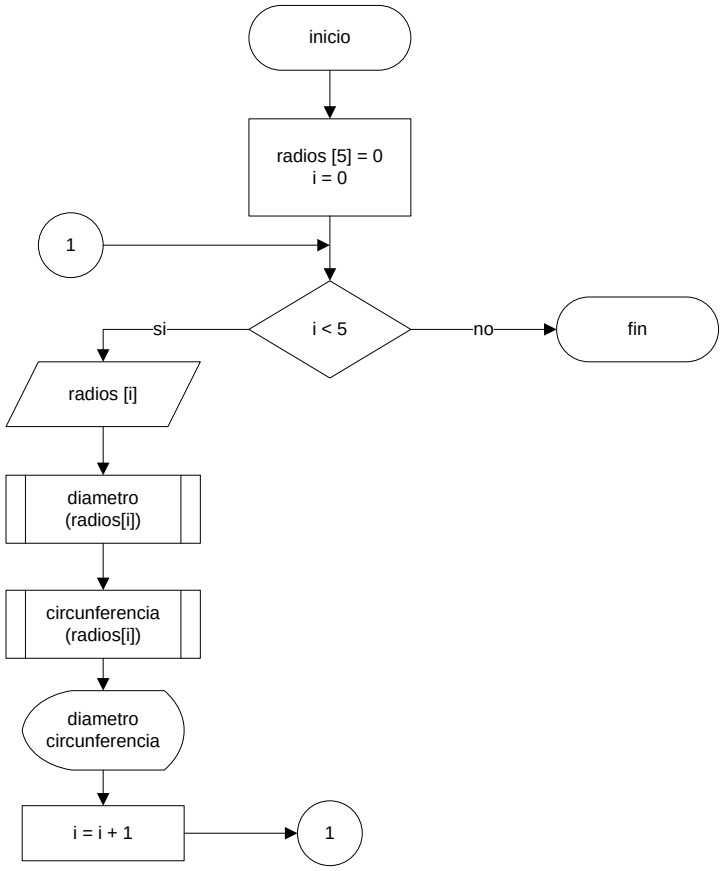
Tercer Ejemplo de automatización de un problema mediante un programa (Estructura de Repetición, Arreglos y Funciones)

Planteamiento del Problema

Se necesita un programa que permita la captura de cinco valores correspondientes a radios de círculos.
Para cada uno de esos cinco valores se requiere que se calcule y muestre en pantalla los siguientes datos del círculo:

1. Diámetro. Se calcula multiplicando el radio por 2
2. Circunferencia. Se calcula multiplicando el diámetro por *pi* (3.1416).

Análisis del problema y algoritmo de solución



Código fuente del programa

```
#include <iostream>
using namespace std;
// Prototipos de función
double diametro (double);
double circunferencia (double);

int main()
{
	double radios [5];
	for (int i = 0; i < 5; i++)
	{	cout << "\nIngrese el radio numero " << i << ": ";
		cin >> radios[i];
		cout << "Diametro del circulo: " << diametro(radios[i]) << endl;
		cout << "Circunferencia: " << circunferencia(radios[i]) << endl;
	}
	system("pause");
}
double diametro (double r)
{	return r * 2; }
double circunferencia (double r)
{	return diametro(r) * 3.1616;}
```

Introducción a la Programación Orientada a Objetos

Definición

La programación Orientada a objetos (POO) es una forma especial de programar, más cercana a como expresaríamos las cosas en la vida real que otros tipos de programación.
Durante años, los programadores se han dedicado a construir aplicaciones muy parecidas que resolvían una y otra vez los mismos problemas. Para conseguir que los esfuerzos de los programadores puedan ser utilizados por otras personas se creó la POO. Que es una serie de normas de realizar las cosas de manera que otras personas puedan utilizarlas y adelantar su trabajo, de manera que consigamos que el código se pueda reutilizar.
La POO no es difícil, pero es una manera especial de pensar, a veces subjetiva de quien la programa, de manera que la forma de hacer las cosas puede ser diferente según el programador.
Aunque podamos hacer los programas de formas distintas, no todas ellas son correctas, lo difícil no es programar orientado a objetos sino programar bien. Programar bien es importante porque así nos podemos aprovechar de todas las ventajas de la POO.

Cómo se piensa en objetos

Pensar en términos de objetos es muy parecido a cómo lo haríamos en la vida real. Por ejemplo vamos a pensar en un coche para tratar de modelizarlo en un esquema de POO. Diríamos que el coche es el elemento principal que tiene una serie de características, como podrían ser el color, el modelo o la marca. Además tiene una serie de funcionalidades asociadas, como pueden ser ponerse en marcha, parar o aparcar.
Pues en un esquema POO el coche sería el objeto, las propiedades serían las características como el color o el modelo y los métodos serían las funcionalidades asociadas como ponerse en marcha o parar.
Por poner otro ejemplo vamos a ver cómo modelizaríamos en un esquema POO una fracción, es decir, esa estructura matemática que tiene un numerador y un denominador que divide al numerador, por ejemplo 3/2.
La fracción será el objeto y tendrá dos propiedades, el numerador y el denominador. Luego podría tener varios métodos como simplificarse, sumarse con otra fracción o número, restarse con otra fracción, etc.
Estos objetos se podrán utilizar en los programas, por ejemplo en un programa de matemáticas harás uso de objetos fracción y en un programa que gestione un taller de coches utilizarás objetos coche. Los programas Orientados a objetos utilizan muchos objetos para realizar las acciones que se desean realizar y ellos mismos también son objetos. Es decir, el taller de coches será un objeto que utilizará objetos coche, herramienta, mecánico, recambios, etc.

Conceptos básicos de la POO

Clases en POO

Las clases son declaraciones de objetos, también se podrían definir como abstracciones de objetos. Esto quiere decir que la definición de un objeto es la clase. Cuando programamos un objeto y definimos sus características y funcionalidades en realidad lo que estamos haciendo es programar una clase. En los ejemplos anteriores en realidad hablábamos de las clases coche o fracción porque sólo estuvimos definiendo, aunque por encima, sus formas.

Propiedades en clases

Las propiedades o atributos son las características de los objetos. Cuando definimos una propiedad normalmente especificamos su nombre y su tipo. Nos podemos hacer a la idea de que las propiedades son algo así como variables donde almacenamos datos relacionados con los objetos.

Métodos en las clases

Son las funcionalidades asociadas a los objetos. Cuando estamos programando las clases las llamamos métodos. Los métodos son como funciones que están asociadas a un objeto.

Objetos en POO

Los objetos son ejemplares de una clase cualquiera. Cuando creamos un ejemplar tenemos que especificar la clase a partir de la cual se creará. Esta acción de crear un objeto a partir de una clase se llama instanciar (que viene de una mala

traducción de la palabra instace que en inglés significa ejemplar). Por ejemplo, un objeto de la clase fracción es por ejemplo 3/5. El concepto o definición de fracción sería la clase, pero cuando ya estamos hablando de una fracción en concreto 4/7, 8/1000 o cualquier otra, la llamamos objeto.

Para crear un objeto se tiene que escribir una instrucción especial que puede ser distinta dependiendo el lenguaje de programación que se emplee, pero será algo parecido a esto:

miCoche = new Coche()

Con la palabra new especificamos que se tiene que crear una instancia de la clase que sigue a continuación. Dentro de los paréntesis podríamos colocar parámetros con los que inicializar el objeto de la clase coche.

En C++, por ejemplo, la sintaxis sería:

Coche miCoche;

Estados en objetos

Cuando tenemos un objeto sus propiedades toman valores. Por ejemplo, cuando tenemos un coche la propiedad color tomará un valor en concreto, como por ejemplo rojo o gris metalizado. El valor concreto de una propiedad de un objeto se llama estado.
Para acceder a un estado de un objeto para ver su valor o cambiarlo se utiliza el operador punto.

miCoche.color = rojo[1]

El objeto es miCoche, luego colocamos el operador punto y por último el nombre de la propiedad a la que deseamos acceder. En este ejemplo estamos cambiando el valor del estado de la propiedad del objeto a rojo con una simple asignación.

Mensajes en objetos

Un mensaje en un objeto es la acción de efectuar una llamada a un método. Por ejemplo, cuando le decimos a un objeto coche que se ponga en marcha estamos pasándole el mensaje “ponte en marcha”.
Para mandar mensajes a los objetos utilizamos el operador punto, seguido del método que deseamos invocar.

miCoche.ponerseEnMarcha()

En este ejemplo pasamos el mensaje ponerseEnMarcha(). Hay que colocar paréntesis igual que cualquier llamada a una función, dentro irían los parámetros.

Otras conceptos a estudiar

En esta materia hay mucho que conocer ya que, hasta ahora, sólo hemos hecho referencia a las cosas más básicas. También existen mecanismos como la herencia y el polimorfismo que son unas de las posibilidades más potentes de la POO.
La herencia sirve para crear objetos que incorporen propiedades y métodos de otros objetos. Así podremos construir unos objetos a partir de otros sin tener que reescribirlo todo.
El polimorfismo sirve para que no tengamos que preocuparnos sobre lo que estamos trabajando, y abstraernos para definir un código que sea compatible con objetos de varios tipos.

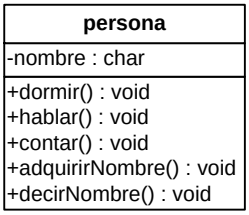
[1] En C++ esta sintaxis es válida cuando las propiedades de los objetos son definidas como públicas. Más adelante en la clase veremos las diferencias entre propiedades públicas y privadas.

Primer Programa Orientado a Objetos

En este primer ejemplo, veremos cómo se define una clase en C++, como se declaran sus propiedades (también conocidas como datos miembro) y métodos (también conocidos como funciones miembro).
También veremos cómo se programan los métodos para establecer lo que hacen una vez que son llamados mediante mensajes.
Y finalmente, veremos cómo se declara un objeto de una clase y se le manipula mediante llamadas a métodos.

Diagrama de clase

El diagrama de clase es una representación semi-gráfica de la clase, que ayuda al programador a visualizar cuales son las propiedades y métodos que contendrá una clase o conjunto de clases en particular.
En un diagrama de clase se pueden representar también relaciones entre clases.
Para el ejemplo que nos ocupa el diagrama de clase es el siguiente:



La clase persona consta de una propiedad o dato miembro, y cinco métodos o funciones.

Código fuente

PrimerEjemplo.cpp

```
#include <iostream>
using namespace std;

// Declaración de la clase
class persona
{
public:
	void dormir();
	void hablar();
	void contar();
	void adquirirNombre();
	void decirNombre();
private:
	char nombre [40];
};

// Declaración de funciones de la clase
void persona::dormir()
{
	cout << "zzzzzzzzz" << endl;
}
void persona::hablar()
{
	cout << "bla bla bla bla" << endl;
}
void persona::contar()
{
	cout << "1, 2, 3, 4, 5..." << endl;
}
void persona::adquirirNombre()
{
	cout << "Soy una persona. Ingrese mi nombre: ";
	cin >> nombre;
}
void persona::decirNombre()
{
	cout << "Mi nombre es: " << nombre << endl;
}

// función principal. Es la porción ejecutable de un programa en C++
int main()
{
	int respuesta = 0;
	// Creando una persona y capturando su nombre
	cout << "Desea crear una persona? 1=Si, 0=No: ";
	cin >> respuesta;
```

```
	if (respuesta == 1)
	{
		persona p;
		p.adquirirNombre();

		// si el usuario lo desea la persona puede decir su nombre
		cout << "Desea que diga mi nombre? 1=Si, 0=No: ";
		cin >> respuesta;
		if (respuesta == 1)
		{
			p.decirNombre();
		}

		// El usuario decide si la persona habla
		cout << "Quiere escucharme hablar? 1=Si, 0=No: ";
		cin >> respuesta;
		if (respuesta == 1)
		{
			p.hablar();
		}

		cout << "Desea que vaya a dormir? 1=Si, 0=No: ";
		cin >> respuesta;
		if (respuesta == 1)
		{
			p.dormir();
		}

		cout << "Desea oirme contar? 1=Si, 0=No: ";
		cin >> respuesta;
		if (respuesta == 1)
		{
			p.contar();
		}
	}
	system("pause");
	return 0;
}
```

Comparativo de Programación Procedural, Modular y Orientada a Objetos: Ejemplo Impuestos

Planteamiento del Problema

El usuario le solicita desarrollar un programa para cálculo de impuestos sobre ventas.

Este programa tiene como propósito calcular el valor que debe pagarse por concepto de impuesto sobre ventas e impuesto al activo neto para un monto dado por el usuario.

Ud. Desarrolla el programa utilizando sus conocimientos adquiridos en Programación Estructurada, para hacer dos versiones, una utilizando un estilo procedural, y la otra utilizando un estilo modular.

Cuando termina el programa y lo muestra al usuario, él le dice que el programa está bien, pero que también necesita que para el mismo monto se calcule el impuesto al activo neto.

A continuación se muestra el desarrollo de este ejercicio utilizando tres técnicas o estilos: **Programación Procedural**, **Programación Modular** y **Programación Orientada a Objetos**. El funcionamiento del programa será prácticamente el mismo en los tres casos, pero observará que utilizando POO el mantenimiento y extensión del programa se va haciendo más fácil.

// Programación Procedural: Cálculo de un impuesto

```
#include <iostream>
using namespace std;

int main()
{       // Definiendo variables
        double tasaISV;
        double impuestoCalculadoISV;
        double valor;

        // Capturando valores
        cout << "Tasa de Impuesto sobre ventas: ";
        cin >> tasaISV;
        cout << "\nValor: ";
        cin >> valor;

        // Cálculos
        impuestoCalculadoISV = valor * tasaISV;

        // Mostrando resultados
        cout << "Para un valor de : " << valor << " y una tasa de: " << tasaISV
                << " el valor de Impuesto sobre Ventas es:"  << impuestoCalculadoISV << endl;

        system("pause");
        return 0;
}
```

// Programación Procedural: Cálculo de dos impuestos

```
#include <iostream>
using namespace std;

int main()
{
        // Definiendo variables
        double tasaISV;
        double impuestoCalculadoISV;
        double tasaIAN;
        double impuestoCalculadoIAN;
        double valor;

        // Capturando valores
        cout << "Tasa de Impuesto sobre ventas: ";
        cin >> tasaISV;
        cout << "Tasa de Impuesto al activo neto: ";
        cin >> tasaIAN;
        cout << "\nValor: ";
        cin >> valor;

        // Cálculos
        impuestoCalculadoISV = valor * tasaISV;
        impuestoCalculadoIAN = valor * tasaIAN;

        // Mostrando resultados
        cout << "Para un valor de : " << valor << " y una tasa de: " << tasaISV
                << " el valor de Impuesto sobre Ventas es:"  << impuestoCalculadoISV << endl;
        cout << "Para un valor de : " << valor << " y una tasa de: " << tasaIAN
                << " el valor de Impuesto al Activo Neto es:"  << impuestoCalculadoIAN << endl;
```

```
        system("pause");
        return 0;
}
```

// Programación Modular: Cálculo de un solo impuesto

```
#include <iostream>
using namespace std;

double calcularImpuesto(double, double);
void capturar(double &, double &);
void imprimir(double, double);

int main()
{
        double tasaISV;
        double valor;

        // Capturando valores
        capturar(tasaISV, valor);
        // Mostrando resultados
        imprimir(tasaISV, valor);

        system("pause");
        return 0;
}

void capturar(double &tISV, double &vlr)
{       cout << "\nTasa de Impuesto sobre ventas: ";
        cin >> tISV;
        cout << "\nValor: ";
        cin >> vlr;
}

void imprimir(double tISV, double vlr)
{
        cout << "Para un valor de : " << vlr << " y una tasa de: " << tISV
                << " el valor de Impuesto sobre Ventas es:"  << calcularImpuesto(vlr, tISV) << endl;
}

double calcularImpuesto(double v, double t)
{       return v * t;
}
```

// Programación Modular: Cálculo de dos impuestos

```
#include <iostream>
using namespace std;

double calcularImpuesto(double, double);
void capturar(double &, double &, double &);
void imprimir(double, double, double);

int main()
{
        double tasaISV;
        double tasaIAN;
        double valor;

        // Capturando valores
        capturar(tasaISV, tasaIAN, valor);

        // Mostrando resultados
        imprimir(tasaISV, tasaIAN, valor);

        system("pause");
        return 0;
}

void capturar(double &tISV, double &tIAN, double &vlr)
{
        cout << "\nTasa de Impuesto sobre ventas: ";
        cin >> tISV;
        cout << "\nTasa de Impuesto al activo neto: ";
        cin >> tIAN;
        cout << "\nValor: ";
        cin >> vlr;
}

void imprimir(double tISV, double tIAN, double vlr)
{

        cout << "Para un valor de : " << vlr << " y una tasa de: " << tISV
                << " el valor de Impuesto sobre Ventas es:"  << calcularImpuesto(vlr, tISV) << endl;
        cout << "Para un valor de : " << vlr << " y una tasa de: " << tIAN
                << " el valor de Impuesto al Activo Neto es:"  << calcularImpuesto(vlr, tIAN) << endl;
}
```

```
double calcularImpuesto(double v, double t)
{
	return v * t;
}
```

// PROGRAMACIÓN ORIENTADA A OBJETOS: Cálculo de un impuesto

```
#include <iostream>
using namespace std;

class impuesto
{
public:
	// Funciones miembro
	void capturar();
	double calcularImpuesto(double);
	void imprimir(double);
	// Datos miembro
	char nombre [25];
	double tasa;
};

int main()
{
	// Definiendo variables y objetos
	impuesto ISV;
	double valor;

	// Capturando valores
	ISV.capturar();
	cout << "Valor: ";
	cin >> valor;

	// Mostrando resultados
	ISV.imprimir(valor);

	system("pause");
	return 0;
}

void impuesto::capturar()
{
	cin.ignore();
	cout << "Nombre de impuesto: ";
	cin.getline(nombre, 25);
	cout << "Tasa: ";
	cin >> tasa;
}

void impuesto::imprimir(double v)
{
	cout << "Para un valor de : " << v << " y una tasa de: " << tasa
		<< " el valor de " << nombre << " es: " << calcularImpuesto(v) << endl;
}

double impuesto::calcularImpuesto(double v)
{
	return v * tasa;
}
```

// PROGRAMACIÓN ORIENTADA A OBJETOS: Cálculo de dos impuestos

```
#include <iostream>
using namespace std;

class impuesto
{
public:
	// Funciones miembro
	void capturar();
	double calcularImpuesto(double);
	void imprimir(double);

	// Datos miembro
	char nombre [25];
	double tasa;
};

int main()
{
	// Definiendo variables y objetos
	impuesto ISV;
	impuesto IAN;
	double valor;

	// Capturando valores
	ISV.capturar();
	IAN.capturar();
```

```
        cout << "Valor: ";
        cin >> valor;

        // Mostrando resultados
        ISV.imprimir(valor);
        IAN.imprimir(valor);

        system("pause");
        return 0;
}

// Las Funciones pertenecen a la clase, no al programa principal…
void impuesto::capturar()
{
        cin.ignore();
        cout << "Nombre de impuesto: ";
        cin.getline(nombre, 25);
        cout << "Tasa: ";
        cin >> tasa;
}

void impuesto::imprimir(double v)
{
        cout << "Para un valor de : " << v << " y una tasa de: " << tasa
                << " el valor de " << nombre << " es: " << calcularImpuesto(v) << endl;
}

double impuesto::calcularImpuesto(double v)
{
        return v * tasa;
}
```

// Salida del programa (Con Programación Procedural y Modular)

```
Tasa de Impuesto sobre ventas: 0.12

Tasa de Impuesto al activo neto: 0.20

Valor: 150000
Para un valor de : 150000 y una tasa de: 0.12 el valor de Impuesto sobre Ventas es:18000
Para un valor de : 150000 y una tasa de: 0.2 el valor de Impuesto al Activo Neto  es:30000
Press any key to continue . . .
```

// Salida del programa (Con Programación Orientada a Objetos)

```
Nombre de impuesto: Impuesto Sobre Ventas
Tasa: 0.12
Nombre de impuesto: Impuesto al Activo Neto
Tasa: 0.20
Valor: 150000
Para un valor de : 150000 y una tasa de: 0.12 el valor de Impuesto Sobre Ventas es: 18000
Para un valor de : 150000 y una tasa de: 0.2 el valor de Impuesto al Activo Neto es: 30000
Press any key to continue . . .
```

Repaso de Funciones y Arreglos

Ejemplo Reglas de Alcance

La porción de programa en donde un identificador se puede utilizar se conoce como **alcance**.

El siguiente ejemplo de código ilustra los cuatro niveles de alcance que puede tener un identificador: de prototipo de función, de función, de bloque o de archivo.

Este ejercicio está basado en la figura 3.12 del libro "Como Programar C++, Deitel & Deitel, Cuarta Edición".

Código fuente

```
// Figura. 3.12: fig03_12.cpp,  Ejemplo de alcance.
#include <iostream>
using namespace std;
void usoLocal( void );        // prototipo de función
void usoStaticLocal( void );  // prototipo de función
void usoGlobal( void );       // prototipo de función
int x = 1;      // variable global

int main()
{
   int x = 5;   // variable local a main
   cout << "x local en el alcance externo de main es " << x << endl;
   { // inicia nuevo alcance
      int x = 7;
      cout << "x local en el alcance interior de main es " << x << endl;
   } // finaliza nuevo alcance
   cout << "x local en el alcance externo de main es " << x << endl;
   usoLocal();       // usoLocal tiene x local
   usoStaticLocal(); // usoStaticLocal tiene x static local
   usoGlobal();      // usoGlobal utiliza x global
   usoLocal();       // usoLocal reinicializa su x local
   usoStaticLocal(); // static local x retiene su valor previo
   usoGlobal();      // x global retiene también su valor
   cout << "\nx local en main es " << x << endl;
   return 0;  // indica terminación exitosa
} // fin de main

// usoLocal reinicializa la variable local x durante cada llamada
void usoLocal( void )
{
   int x = 25;  // inicialia cada vez que se llama a usoLocal
   cout << endl << "x local es " << x  << " al entrar a usoLocal" << endl;
   ++x;
   cout << "x local es " << x  << " al salir de usoLocal" << endl;
} // fin de la función usoLocal

// usoStaticLocal inicializa a la variable static local x sólo la primera vez que se
// llama a la función; el valor de x se guarda entre las llamadas a esta función
void usoStaticLocal( void )
{
   // se inicializa la primera vez que se llama a usoStaticLocal.
   static int x = 50;
   cout << endl << "local static x es " << x  << " al entrar a usoStaticLocal" << endl;
   ++x;
   cout << "local static x es " << x  << " al salir de usoStaticLocal" << endl;
} // fin de la función usoStaticLocal

// usoGlobal modifica la variable global x durante cada llamada
void usoGlobal( void )
{
   cout << endl << "x global es " << x  << " al entrar a usoGlobal" << endl;
   x *= 10;
   cout << "x global es " << x  << " al salir de usoGlobal" << endl;
} // fin de la función usoGlobal
```

Salida del programa

```
x local en el alcance externo de main es 5
x local en el alcance interior de main es 7
x local en el alcance externo de main es 5

x local es 25 al entrar a usoLocal
x local es 26 al salir de usoLocal

local static x es 50 al entrar a usoStaticLocal
local static x es 51 al salir de usoStaticLocal

x global es 1 al entrar a usoGlobal
x global es 10 al salir de usoGlobal
```

```
x local es 25 al entrar a usoLocal
x local es 26 al salir de usoLocal

local static x es 51 al entrar a usoStaticLocal
local static x es 52 al salir de usoStaticLocal

x global es 10 al entrar a usoGlobal
x global es 100 al salir de usoGlobal
x local en main es 5
Press any key to continue
```

Ejemplo Funciones que no toman, ni devuelven Argumentos

Las funciones que no *toman* o *reciben* argumentos (o con listas de parámetros vacías), son aquellas que no necesitan un valor externo (provisto como argumento) para cumplir su propósito.
Existe también el caso en que no se requiera que la función devuelva al programa que la invoca un valor específico. En este caso se dice que no *devuelve* argumento.
En programación orientada a objetos se utiliza mucho este tipo de funciones, ya que todas las acciones que se realizan sobre los objetos se hacen mediante funciones, incluso cuando los datos del objeto no serán modificados.
Este ejercicio está basado en la figura 3.18 del libro "Como Programar C++, Deitel & Deitel, Cuarta Edición".

Código fuente

```
// Figura. 3.18: fig03_18.cpp
// Funciones que no toman argumentos.
#include <iostream>
using std::cout;
using std::endl;

void funcion1();        // prototipo de la función
void funcion2( void );  // prototipo de la función

int main()
{
   funcion1();  // llama a la funcion1 sin argumentos
   funcion2();  // llama a la funcion2 sin argumentos
   return 0;     // indica terminación exitosa
} // fin de main

// la funcion1 utiliza una lista de parámetros vacía para especificar que
// la función no recibe argumentos
void funcion1()
{
   cout << "La funcion1 no toma argumentos" << endl;
} // fin de funcion1

// La funcion2 utiliza la lista de parámetros void para especificar que
// la función no recibe argumentos
void funcion2( void )
{
   cout << "La funcion2 tampoco toma argumentos" << endl;
} // fin de funccion2
```

Salida

```
La funcion1 no toma argumentos
La funcion2 tampoco toma argumentos
Press any key to continue
```

Ejemplo Parámetros por valor y por referencia

Existen dos maneras de pasar argumentos a una función:

- Paso por valor
- Paso por referencia.

Paso por valor: Cuando un argumento se pasa por valor, se hace una copia del argumento y se pasa a la función. Los cambios que se hagan a la copia no afectan al valor original de la variable llamada.

La desventaja del paso por valor es que, si se pasa un gran elemento de datos, copiar ese dato puede tomar una considerable cantidad de tiempo y espacio en memoria

Paso por referencia: La llamada le da a la función invocada la habilidad de acceder directamente a los datos y modificar esos datos si la función invocada lo desea.

El paso por referencia es bueno por razones de rendimiento, pero debilita la seguridad porque la función invocada puede corromper los datos de la función que invoca.

Este ejercicio está basado en la figura 3.20 del libro "Como Programar C++, Deitel & Deitel, Cuarta Edición".

Código fuente

```
// Figura. 3.20: fig03_20.cpp
// Comparación entre el paso por valor y el paso por referencia
// mediante referencias.
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

int cuadradoPorValor( int );         // prototipo de función
void cuadradoPorReferencia( int & ); // prototipo de función

int main()
{
   int x = 2;
   int z = 4;

   // demuestra cuadradoPorValor
   cout << "x = " << x << " antes de cuadradoPorValor\n";
   cout << "Value devuelto por cuadradoPorValor: "
        << cuadradoPorValor( x ) << endl;
   cout << "x = " << x << " despues de cuadradoPorValor\n" << endl;

   // demuestra cuadradoPorReferencia
   cout << "z = " << z << " antes de cuadradoPorReferencia" << endl;
   cuadradoPorReferencia( z );
   cout << "z = " << z << " despues de cuadradoPorReferencia" << endl;

   return 0;
} // fin de main

// cuadradoPorValor multiplica el número por sí mismo, almacena
// el resultado en número y devuelve el nuevo valor de número
int cuadradoPorValor( int numero )
{
   return numero *= numero;  // argumento de la llamada no modificado

} // fin de la función cuadradoPorValor

// cuadradoPorReferencia multiplica numeroRef por sí mismo y
// almacena el resultado en la variable a la cual se refiere numeroRef
// en la función main
void cuadradoPorReferencia( int &numeroRef )
{
   numeroRef *= numeroRef;   // argumento de lallamada modificado

} // fin de la función cuadradoPorReferencia
```

Salida

```
x = 2 antes de cuadradoPorValor
Value devuelto por cuadradoPorValor: 4
x = 2 despues de cuadradoPorValor

z = 4 antes de cuadradoPorReferencia
z = 16 despues de cuadradoPorReferencia
Press any key to continue
```

Ejemplo Uso de Argumentos Predeterminados

No es extraño para un programa invocar una función de manera repetida con el mismo valor de argumento para un parámetro en particular.

Algunas reglas que rigen el uso de argumentos predeterminados son:

- Cuando el programa omite un argumento predeterminado en una llamada de función, el compilador reescribe la llamada e inserta el valor predeterminado de dicho argumento para que se para que se pase a la llamada de función.
- Los argumentos predeterminados deben estar a la extrema derecha de la lista de parámetros.
- Debe especificarse con la primera ocurrencia del nombre de la función es decir, en el prototipo de función.
- Los valores predeterminados pueden ser: constantes, variables globales o llamadas a funciones.

Este ejercicio está basado en la figura 3.23 del libro "Como Programar C++, Deitel & Deitel, Cuarta Edición".

Código fuente

```
// Figura 3.23: fig03_23.cpp
       // Uso de argumentos predeterminados.
       #include <iostream>

       using std::cout;
       using std::endl;

       // prototipo de la función que especifica argumentos predeterminados
       int volumenCaja( int longitud = 1, int ancho = 1, int altura = 1 );

       int main()
       {
          // sin argumentos--utilice valores predeterminados para todas las dimensiones
          cout << "El volumen predeterminado de la caja es: " << volumenCaja();

          // especifique la longitud, ancho y altura predeterminados
          cout << "\n\nEl volumen de una caja de longitud 10,\n"
               << "ancho 1 y altura 1 es: " << volumenCaja( 10 );

          // epecifique la longitud, ancho y altura predeterminados
          cout << "\n\nEl volumen de una caja de longitud 10,\n"
               << "ancho 5 y altura 1 es: " << volumenCaja( 10, 5 );

          // especifica todos los argumentos
          cout << "\n\nEl volumen de una caja de longitud 10,\n"
               << "ancho 5 y altura 2 es: " << volumenCaja( 10, 5, 2 )
               << endl;

          return 0;  // indica terminación exitosa

       } // fin de main

       // la función volumenCaja calcula el volumen de una caja
       int volumenCaja( int longitud, int ancho, int altura )
       {
          return longitud * ancho * altura;

       } // fin de la función volumenCaja
```

Salida

```
El volumen predeterminado de la caja es: 1

El volumen de una caja de longitud 10,
ancho 1 y altura 1 es: 10

El volumen de una caja de longitud 10,
ancho 5 y altura 1 es: 50

El volumen de una caja de longitud 10,
ancho 5 y altura 2 es: 100
Press any key to continue
```

Estructuras y Clases

Ejemplo de Estructura

La estructura es una primera aproximación a la programación orientada a objetos.
Se define una estructura cuando se necesitan crear identificadores (variables, constantes, etc.) compuestos por más de un dato.
Por ejemplo: Definiríamos una estructura llamada dirección si necesitáramos variables para almacenar direcciones y las mismas estuvieran compuestas por: calle, avenida, ciudad, y número de casa, por ejemplo.
Este ejercicio está basado en la figura 6.01 del libro "Como Programar C++, Deitel & Deitel, Cuarta Edición".

Código Fuente

```
// Crea una estructura, establece sus miembros, y la imprime.
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

#include <iomanip>
using std::setfill;
using std::setw;

// definición de la estructura Tiempo
struct Tiempo
{
   int hora;      // 0-23 (formato de reloj de 24 horas)
   int minuto;    // 0-59
   int segundo;   // 0-59
}; // fin de la estructura Tiempo

void imprimeUniversal( const Tiempo & );   // prototipo
void imprimeEstandar( const Tiempo & );    // prototipo

int main()
{
       // Declarando variable del nuevo tipo Tiempo
       Tiempo horaCena;

       // Capturando valores para los datos miembro
       cout << "Ingrese la hora de la cena" << endl;
       cout << "Hora: ";
       cin >> horaCena.hora;    // establece el valor del miembro hora de horaCena
       cout << "Minutos: ";
       cin >> horaCena.minuto;   // establece el valor del miembro minuto de horaCena
       cout << "Segundos: ";
       cin >> horaCena.segundo;  // establece el valor del miembro segundo de horaCena

       // Imprime la hora utilizando las funciones
       cout << "La cena se servira a las ";
       imprimeUniversal( horaCena );
       cout << " en hora universal,\nla cual es ";
       imprimeEstandar( horaCena );
       cout << " en hora estandar.\n";
       cout << endl;

       system("PAUSE");
       return 0;
} // fin de main

// imprime la hora en el formato universal de tiempo
void imprimeUniversal( const Tiempo &t )
{
   cout << setfill( '0' ) << setw( 2 ) << t.hora << ":"
        << setw( 2 ) << t.minuto << ":"
        << setw( 2 ) << t.segundo;
} // fin de la función imprimeUniversal

// imprime la hora en formato estándar de tiempo
void imprimeEstandar( const Tiempo &t )
{
   cout << ( ( t.hora == 0 || t.hora == 12 ) ?
             12 : t.hora % 12 ) << ":" << setfill( '0' )
        << setw( 2 ) << t.minuto << ":"
        << setw( 2 ) << t.segundo
        << ( t.hora < 12 ? " AM" : " PM" );
} // fin de la función imprimeEstandar
```

Salida

```
Ingrese la hora de la cena
Hora: 22
Minutos: 15
Segundos: 09
La cena se servira a las 22:15:09 en hora universal,
la cual es 10:15:09 PM en hora estandar.

Presione una tecla para continuar . . .
```

Ejemplo de Clase vrs. Estructura

La diferencia fundamental entre una estructura y una clase es que para la estructura solo se definen datos, no funciones. Si existen funciones que trabajan sobre los datos de la estructura, las mismas son independientes y necesitan recibir como parámetros los valores de los datos de la estructura para poder trabajar con ellos.

La necesidad de enviar dichos parámetros es señal característica de que las funciones no tienen una relación directa con la estrcutura.

En una clase, por el contrario, las funciones son parte de la misma, y reflejan comportamientos de la clase u operaciones que pueden realizar sobre los datos de la clase. Al estar asociadas a la clase estas funciones no necesitan recibir como parámetros los datos propios de ella, sino, únicamente, aquellos valores externos que sirvan para realizar correctamente su propósito.

En el siguiente ejemplo se desarrolla el mismo programa anterior, con la diferencia de que en lugar de resolverlo con una estructura se utiliza una clase.

Este ejercicio está basado en la figura 6.03 del libro “Como Programar C++, Deitel & Deitel, Cuarta Edición”.

Código Fuente

```
#include <iostream>
using std::cout;
using std::endl;
using std::cin;
#include <iomanip>
using std::setfill;
using std::setw;

// Definición del tipo de dato abstracto (ADT) Tiempo
class Tiempo
{
public:
   Tiempo();                          // constructor
   void estableceHora( int, int, int ); // establece hora, minuto, segundo
   void capturarDatos();
   void imprimeUniversal();         // imprime el tiempo en formato universal
   void imprimeEstandar();          // imprime el tiempo en formato estándar
private:
   int hora;     // 0 - 23 (formato de reloj de 24 horas)
   int minuto;   // 0 - 59
   int segundo;   // 0 - 59
}; // fin de la clase Tiempo

// el constructor Tiempo inicializa cada dato miembro en cero y
// garantiza que los objetos Tiempo comiencen en un estado consistente
Tiempo::Tiempo()
{
   hora = minuto = segundo = 0;
} // fin del constructor Tiempo

// establece un nuevo valor para Tiempo de acuerdo con la hora universal, realiza la validación
// de los valores de datos y establece los valores no válidos en cero
void Tiempo::estableceHora( int h, int m, int s )
{
   hora = ( h >= 0 && h < 24 ) ? h : 0;
   minuto = ( m >= 0 && m < 60 ) ? m : 0;
   segundo = ( s >= 0 && s < 60 ) ? s : 0;
} // fin de la función estableceHora

// función para capturar datos de hora
void Tiempo::capturarDatos()
{
        int hr, mn, sg = 0;
        cout << "Hora: ";
        cin >> hr;
        cout << "Minutos: ";
        cin >> mn;
        cout << "Segundos: ";
        cin >> sg;
        estableceHora(hr, mn, sg);
} // fin de la funcion capturarDatos
```

```
// imprime Tiempo en formato universal
void Tiempo::imprimeUniversal()
{
   cout << setfill( '0' ) << setw( 2 ) << hora << ":" << setw( 2 ) << minuto << ":"
        << setw( 2 ) << segundo;
} // fin de la función imprimeUniversal

// imprime Tiempo en formato estándar
void Tiempo::imprimeEstandar()
{
   cout << ( ( hora == 0 || hora == 12 ) ? 12 : hora % 12 )
        << ":" << setfill( '0' ) << setw( 2 ) << minuto  << ":" << setw( 2 ) << segundo
        << ( hora < 12 ? " AM" : " PM" );
} // fin de la función imprimeEstandar

int main()
{
   Tiempo horaCena;  // crea la instancia del objeto t de la clase Tiempo

   // Capturando los datos de la hora de la cena
   horaCena.capturarDatos();

   // Imprimiendo los datos de la cena
   // Imprime la hora utilizando las funciones
        cout << "La cena se servira a las ";
        horaCena.imprimeUniversal();
        cout << " en hora universal,\nla cual es ";
        horaCena.imprimeEstandar();
        cout << " en hora estandar.\n";
        cout << endl;

   system ("pause");
   return 0;
} // fin de main
```

Salida

```
Hora: 20
Minutos: 15
Segundos: 59
La cena se servira a las 20:15:59 en hora universal,
la cual es 8:15:59 PM en hora estandar.

Presione una tecla para continuar . . .
```

Validación de datos

Ejemplo de validación de datos en una clase

Cuando se alimentan de valores los datos de un objeto es importante asegurarse de que dichos datos son válidos.
Si la validación de los datos se coloca en las funciones de la clase, siempre estaremos seguros de que los objetos que instanciemos de ella serán correctamente validados.
A continuación se muestra un ejemplo en el que, para una clase, se definen funciones que permitan capturar y mostrar en pantalla datos.
La clase a definir se llama usuario. La clase usuario tiene dos datos miembro: Nombre de Identificación del Usuario (conocido como ID) y el Número de Identificación Personal (conocido como PIN). Para efectos de este ejemplo, se considerará válido un ID que tenga una longitud mínima de 6 caracteres; y en el caso del PIN el valor válido será cualquier número mayor que cero.
La primera versión del ejemplo no utiliza una función para validar datos. Al utilizar la clase en un programa y crear objetos con ella, veremos que es posible establecer valores inválidos en los datos.
En la segunda versión se le incorpora una función pero permita validar antes de establecer los datos. Además se modifica la función con la que se capturan los datos en pantalla, obligando a que cuando se modifiquen los datos se tenga que pasar sí o sí por la función que valida.

Ejemplo Clase usuario sin validación

```
// Ejemplo de clase usuario, sin validar datos

#include <iostream>
using namespace std;

// Declaración de la clase
class usuario
{
private:
	char id [15];
	int pin;
public:
	void capturarDatos();
	void imprimirDatos();
};

// Declaración de las funciones de la clase

// Con esta función se capturan en pantalla los datos para objetos de la clase usuario
void usuario::capturarDatos()
{
	cout << "Ingrese el ID de Usuario: ";
	cin.getline(id, 15);
	cout << "Ingrese Numero de Identificación Personal (PIN): ";
	cin >> pin;
}

// Con esta función se muestran en pantalla los datos de un objeto de la clase usuario
void usuario::imprimirDatos()
{
	cout << "ID: " << id << endl;
	cout << "PIN: " << pin << endl;
}

// Probando la clase
int main()
{
	usuario u;
	u.capturarDatos();
	u.imprimirDatos();

	system("pause");
}
```

Ejemplo Clase usuario con validación

```
// Ejemplo de clase usuario, validando datos

#include <iostream>
using namespace std;

// Declaración de la clase
class usuario
```

```
{
private:
	char id [15];
	int pin;
public:
	void capturarDatos();
	void imprimirDatos();

	// La función establecer datos se encargará de validar los datos que se
	// ingresan para los usuarios, asegurándose así de que siempre se ingresan
	// valores válidos en los mismos
	void establecerDatos(char [15], int);
};

// Declaración de las funciones de la clase

// Con esta función se capturan en pantalla los datos para objetos de la clase usuario
void usuario::capturarDatos()
{
	char i [15];
	int p = 0;

	cout << "Ingrese el ID de Usuario: ";
	cin.getline(i, 15);
	cout << "Ingrese Numero de Identificación Personal (PIN): ";
	cin >> p;

	establecerDatos(i, p);
}

// Con esta función se muestran en pantalla los datos de un objeto de la clase usuario
void usuario::imprimirDatos()
{
	cout << "ID: " << id << endl;
	cout << "PIN: " << pin << endl;
}

void usuario::establecerDatos(char i [15], int p)
{
	if (strlen(i) < 6)
	{
		cout << "ID de usuario muy corto" << endl;
		strcpy_s(id, "desconocido");
	}
	else
	{
		strcpy_s(id, i);
	}

	if (p <= 0)
	{
		cout << "El PIN debe ser mayor que cero" << endl;
		pin = 1;
	}
	else
	{
		pin = p;
	}
}

// Probando la clase
int main()
{
	usuario u;
	u.capturarDatos();
	u.imprimirDatos();

	system("pause");
}
```

Constructores

Ejemplo Constructores con Valores Predeterminados

En este ejemplo se define una clase Tiempo con un comportamiento especial.
En los programas cliente de esta clase, los objetos de tipo tiempo se inicializan con los valores 12:00:00.
Pero además de eso, es posible, al declarar objetos de tipo Tiempo indicar valores iniciales para los datos miembro, sin necesidad de invocar a la función establecer. Esto es posible porque la función constructor de la clase Tiempo tiene valores predeterminados.
También es posible inicializar objetos de tipo Tiempo indicando solo la hora; solo la hora y los minutos o; por supuesto, hora, minutos y segundos.
Este ejercicio está basado en la figura 6.03 del libro "Como Programar C++, Deitel & Deitel, Cuarta Edición".

Código fuente

```
#include <iostream>
using std::cin;
using std::cout;
using std::endl;
#include <iomanip>
using std::setfill;
using std::setw;
// Definición del tipo de dato abstracto (ADT) Tiempo
class Tiempo {
public:
   Tiempo( int = 12, int = 0, int = 0);  // constructor con valores predeterminados
   void estableceHora( int, int, int ); // establece hora, minuto, segundo
   void imprimeUniversal();         // imprime el tiempo en formato universal
   void imprimeEstandar();          // imprime el tiempo en formato estándar
   void capturarHora();
private:
   int hora;     // 0 - 23 (formato de reloj de 24 horas)
   int minuto;   // 0 - 59
   int segundo;   // 0 - 59
}; // fin de la clase Tiempo

// el constructor Tiempo inicializa cada dato miembro en cero y
// garantiza que los objetos Tiempo comiencen en un estado consistente
Tiempo::Tiempo(int h, int m, int s)
{   estableceHora(h, m, s);
} // fin del constructor Tiempo

// establece un nuevo valor para Tiempo de acuerdo con la hora universal, realiza la validación
// de los valores de datos y establece los valores no válidos en cero
void Tiempo::estableceHora( int h, int m, int s )
{
   hora = ( h >= 0 && h < 24 ) ? h : 0;
   minuto = ( m >= 0 && m < 60 ) ? m : 0;
   segundo = ( s >= 0 && s < 60 ) ? s : 0;
} // fin de la función estableceHora

void Tiempo::capturarHora()
{
        int h, m, s = 0;
        cout << "Hora: ";
        cin >> h;
        cout << "Minutos: ";
        cin >> m;
        cout << "Segundos: ";
        cin >> s;
        estableceHora(h, m, s);
}

// imprime Tiempo en formato universal
void Tiempo::imprimeUniversal()
{
   cout << setfill( '0' ) << setw( 2 ) << hora << ":" << setw( 2 ) << minuto << ":"
        << setw( 2 ) << segundo;
} // fin de la función imprimeUniversal

// imprime Tiempo en formato estándar
void Tiempo::imprimeEstandar()
{
   cout << ( ( hora == 0 || hora == 12 ) ? 12 : hora % 12 ) << ":" << setfill( '0' )
   << setw( 2 ) << minuto << ":" << setw( 2 ) << segundo << ( hora < 12 ? " AM" : " PM" );
```

```
} // fin de la función imprimeEstandar

int main()
{
   Tiempo horaLevantarse;  // crea un objeto de la clase Tiempo
   cout << "A que hora se levanta Ud.?: " << endl;
   horaLevantarse.capturarHora();
   cout  << "\nen Hora universal: ";
   horaLevantarse.imprimeUniversal();  // 00:00:00
   cout << "\nen Hora estandar: ";
   horaLevantarse.imprimeEstandar();   // 12:00:00 AM
   cout << endl;

   // Ahora probando con constructores predeterminados
   Tiempo horaDesayuno(7, 15, 30);
   Tiempo horaAlmuerzo(12, 30);
   Tiempo horaCena(18);
   cout << "\nEl desayuno se sirve a las: ";
   horaDesayuno.imprimeEstandar();
   cout << "\nEl almuerzo se sirve a las: ";
   horaAlmuerzo.imprimeEstandar();
   cout << "\nLa cena se sirve a las : ";
   horaCena.imprimeEstandar();
   cout << endl;
   system ("pause");
   return 0;
} // fin de main
```

Salida

```
A que hora se levanta Ud.?:
Hora: 5
Minutos: 45
Segundos: 0

en Hora universal: 05:45:00
en Hora estandar: 5:45:00 AM

El desayuno se sirve a las: 7:15:30 AM
El almuerzo se sirve a las: 12:30:00 PM
La cena se sirve a las : 6:00:00 PM
Press any key to continue . . .
```

Más ejercicios básicos sobre clases

Ejemplo de clase utilizando cadenas de caracteres

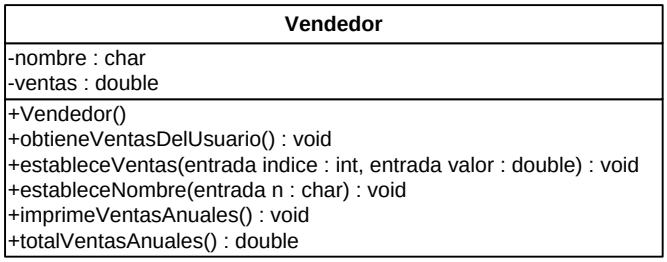
Planteamiento del Problema

Se necesita de un programa que sirva para capturar el nombre y los valores de las ventas mensuales de un vendedor. Son 12 valores, uno por cada mes del año.

El programa debe repetirse para tantos vendedores como el usuario desea ingresar. Después de ingresar e imprimir los datos de un vendedor debe preguntarse al usuario si desea continuar o salir del programa. Cuando el usuario seleccione salir debe imprimirse un gran total de las ventas registradas en toda la corrida del programa.

Se requiere realizar el programa utilizando POO, manejando en archivos separados la declaración de la clase, la declaración de las funciones de la clase y el programa cliente que realice lo requerido.

Diagrama de Clase



Código fuente

Vendedor.h

```
// Definición de la clase Vendedor.
// Funciones miembro definidas en Vend.cpp.
#ifndef VEND_H
#define VEND_H
class Vendedor {
public:
   Vendedor();                        // constructor
   void obtieneVentasDelUsuario(); // introduce ventas desde el teclado
   void estableceVentas( int, double ); // establece las ventas por mes
   void imprimeVentasAnuales();   // suma e imprime las ventas
   void estableceNombre( char [40]);
   double totalVentasAnuales();        // función de utilidad
private:
   double ventas[ 12 ];           // 12 cantidades mensuales de ventas
   char nombre [40];
}; // fin de la clase Vendedor
#endif
```

Vendedor.cpp

```
// Funciones miembro para la clase Vendedor.
#include <iostream>
using namespace std;
#include <iomanip>
using std::setprecision;
// incluye la definición de la clase Vendedor desde Vend.h
#include "vendedor.h"

// inicializa los elementos del arreglo ventas en 0.0
Vendedor::Vendedor()
{
    // Inicializa las ventas en cero
       for ( int i = 0; i < 12; i++ )
      ventas[ i ] = 0.0;
    // Inicializa el nombre en blanco
       strcpy_s(nombre, " ");
} // fin del constructor Vendedor

// obtiene 12 cifras de ventas del usuario desde el teclado
void Vendedor::obtieneVentasDelUsuario()
{
    char n [40];
       double cantVentas;
   // Introduce el nombre del vendedor
   cout << "Nombre del Vendedor: ";
   cin.ignore();
```

```
   cin.getline(n, 40);
   for ( int i = 1; i <= 12; i++ )
   {
      cout << "Introduzca el monto de ventas del mes " << i << ": ";
      cin >> cantVentas;
      estableceVentas( i, cantVentas );
         estableceNombre(n);
   } // fin de for
} // fin de la función obtieneVentasDelUsuario

// establece una de las 12 cifras de ventas mensuales; la función resta
// uno al valor del mes para establecer el subíndice apropiado en el arreglo ventas.
void Vendedor::estableceVentas( int mes, double monto )
{
   // evalúa la validez de los valores del mes y del monto
   if ( mes >= 1 && mes <= 12 && monto > 0 )
      ventas[ mes - 1 ] = monto; // ajuste para los subíndices 0-11
   else // valor de mes o monto inválido
      cout << "Valor de mes o de ventas no valido" << endl;
} // fin de la función estableceVentas

// Establece el nombre del vendedor
void Vendedor::estableceNombre(char n [40])
{      strcpy_s(nombre, n);
}
// imprime las ventas totales anuales (con la ayuda de la función de utilidad)
void Vendedor::imprimeVentasAnuales()
{
   cout << setprecision( 2 ) << fixed
           << "\nLas ventas anuales totales del vendedor "
        << nombre << " son: $"
        << totalVentasAnuales() << endl; // llamada a la función de utilidad
} // fin de la función imprimeVentasAnuales

// función privada de utilidad para sumar las ventas anuales
double Vendedor::totalVentasAnuales()
{
   double total = 0.0;             // inicializa total
   for ( int i = 0; i < 12; i++ )  // suma los resultados de las ventas
      total += ventas[ i ];
   return total;
} // fin de la función totalVentasAnuales
```

ejemploVendedores.cpp

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

int main()
{      // Programa para capturar datos de un número indefinido de vendedores
       Vendedor v;
       int salir = 0;
       double granTotal = 0;
       while (salir >= 0)
       {      cout << "Ingrese datos de ventas del vendedor: " << endl;
              v.obtieneVentasDelUsuario();
              v.imprimeVentasAnuales();
              granTotal += v.totalVentasAnuales();
              cout << "\nDigite -1 para salir, o 1 para continuar: ";
              cin >> salir;
       }
       // Imprime el gran total de ventas registradas
       cout << "Gran total de ventas registradas: $" << granTotal << endl;
       system ("pause");
       return 0;
}
```

Salida

```
Ingrese datos de ventas del vendedor:
Nombre del Vendedor: James Cameron
Introduzca el monto de ventas del mes 1: 30000
Introduzca el monto de ventas del mes 2: 50000
Introduzca el monto de ventas del mes 3: 100000
Introduzca el monto de ventas del mes 4: 80000
Introduzca el monto de ventas del mes 5: 75000
Introduzca el monto de ventas del mes 6: 25000
Introduzca el monto de ventas del mes 7: 90000
Introduzca el monto de ventas del mes 8: 120000
Introduzca el monto de ventas del mes 9: 150000
Introduzca el monto de ventas del mes 10: 110000
Introduzca el monto de ventas del mes 11: 200000
Introduzca el monto de ventas del mes 12: 130000

Las ventas anuales totales del vendedor James Cameron son: $1160000.00

Digite -1 para salir, o 1 para continuar: -1
Gran total de ventas registradas: $1160000.00
```

```
Press any key to continue . . .
```

Ejemplo de clases utilizando funciones para manipulación de cadenas de caracteres

Planteamiento del Problema

Se requiere de un program que permita capturar e imprimir los datos de dos cuentas de correo electronico. Cada dato de la cuenta de correo (Id, dominio y password) dede digitarse por separado.
Al imprimirse la cuenta los datos de ID y dominio deben aparecer unidos. Por ejemplo: zelaya.luis@gmail.com.
Debe validarse el password, solicitando que sea digitado dos veces y debe coincidir en ambas. Si este caso se da, no se establecen los datos de la cuenta de correo.

Código fuente

Email.h

```
#ifndef clase_email
#define clase_email
// La clase cEMail sirve para definir objetos de tipo correo electrónico
// para los cuales se guardarán como miembros: el ID de usuario, el dominio
// y el password.

class cEMail
{
private:
	char idCuenta [25];
	char dominio [25];
	char pass [25];
public:
	// El constructor inicializa cada objeto de tipo cEmail con valores validos
	cEMail();

	// Las funciones establecer se utilizan para modificar los valores de los datos miembro
	// de forma segura.
	// Estas funciones reciben parametros o argumentos, porque necesitan saber
	// cuales valores modificar el contenido de los datos miembros
	// estos valores son enviados desde los programas cliente de la clase, o desde otras
	// funciones de las misma clase
	void establecerEMail(char [25], char [25], char [25]);

	// Esta funcion se utiliza para que desde los programas cliente de la clase puedan digitarse
	// valores que modifiquen el contenido de los datos miembro de la clase
	void capturarEMail();

	// Esta funcion se utiliza para que desde los programas cliente de la clase se puedan
	// visualizar los valores actuales de los datos miembros, asi como otros valores derivados
	// de dichos datos miembro
	void imprimirEMail();
};
#endif
```

Email.cpp

```
// Archivo de Funciones de Clase EMail
#include <iostream>
using namespace std;
#include "eMail.h"

cEMail::cEMail()
{	char id [25];
	char dom [25];
	char pas [25];
	strcpy_s(id, "Desconocido");
	strcpy_s(dom, "Desconocido");
	strcpy_s(pas, "");
	establecerEMail(id, dom, pas);
}

void cEMail::establecerEMail(char c [25], char d [25], char p [25])
{	strcpy_s(idCuenta, c);
	strcpy_s(dominio, d);
	strcpy_s(pass, p);
}

void cEMail::capturarEMail()
{	// Una correcta funcion de captura realiza los pasos siguientes:
	// 1. Se definen variables locales para capturar los datos en ellas.
	char id [25];
```

```
        char dom [25];
        char pas1 [25];
        char pas2 [25];

        // 2. Se capturan los datos en pantalla
        cin.ignore();
        cout << "Id de Usuario: ";
        cin.getline(id, 25);
        cout << "Dominio: ";
        cin.getline(dom, 25);
        cout << "Password: ";
        cin.getline(pas1, 25);
        cout << "Digite nuevamente el password: ";
        cin.getline(pas2, 25);

        // 3. Se validan los datos ingresados, si se requiere
        // En este caso se requiere validad que los valores digitados en pas1 y pas2 coincidan
        if (strcmp(pas1, pas2) != 0)
        {               cout << "Error, los passwords no coinciden" << endl;
        }
        else
        {               establecerEMail(id, dom, pas1);
        }
}

void cEMail::imprimirEMail()
{       cout << "Cuenta de correo: " << idCuenta << "@" << dominio << endl;
}
```

```
// ejemploEMail.cpp
#include <iostream>
using namespace std;
#include "eMail.h"

int main()
{
        // Se trabajara con dos cuentas distintas de correo electronico
        // por eso se define un arreglo de dos objetos de tipo cEMail
        cEMail correos [2];

        // Capturando los datos de direcciones de correo
        for (int i = 0; i < 2; i++)
        {       cout << "Presione ENTER para ingresar datos de correo numero " << i + 1 << endl;
                correos[i].capturarEMail();
                cout << endl;
        }

        // Mostrando los datos de forma individual
        for (int i = 0; i < 2; i++)
        {       correos[i].imprimirEMail();
                cout << endl;
        }

        system("pause");
        return 0;
}
```

Salida

```
Presione ENTER para ingresar datos de correo numero 1

Id de Usuario: lzelaya
Dominio: yahoo.com
Password: luis
Digite nuevamente el password: luis

Presione ENTER para ingresar datos de correo numero 2

Id de Usuario: lfzi
Dominio: gmail
Password: luis
Digite nuevamente el password: fernando
Error, los passwords no coinciden

Cuenta de correo: lzelaya@yahoo.com

Cuenta de correo: Desconocido@Desconocido

Press any key to continue . . .
```

SEGUNDO PARCIAL

Funciones Set/Get

Ejemplo de Clase Tiempo Completa

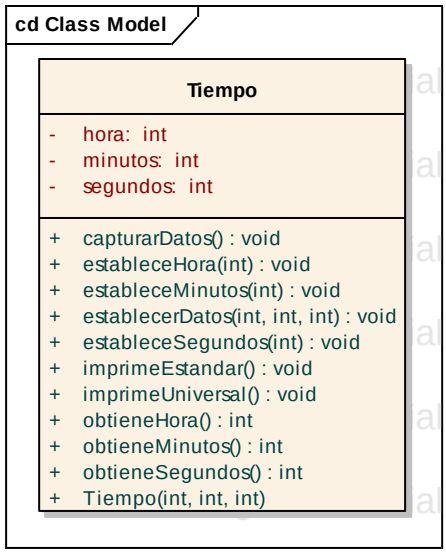
Explicación del Ejemplo

Este ejercicio tiene como finalidad perfeccionar la clase Tiempo que ya ha sido trabajada en ejercicios previos de esta clase, incorporándole la funciones establecer (set) y obtener (get) para cada dato miembro de la clase.
En este ejemplo se muestra todo el código fuente de la clase Tiempo con todos los datos miembro y funciones que se especifican en el diagrama de clase, que muestra en esta página, y luego se incluye un programa cliente que demuestre el uso de cada una de esas funciones.

Puntos relevantes mostrados en este ejemplo:

- Definición, declaración y uso de funciones set y get.

Diagrama de Clase Tiempo



Código fuente

// tiempoSetGet.h

```
// Fig. 6.18: tiempoSetGet.h
// Declaración de la clase Tiempo.
// Funciones miembro definidas en tiempo3.cpp

// previene la inclusión múltiple del archivo de encabezado
#ifndef TIEMPO3_H
#define TIEMPO3_H

class Tiempo {
public:
        // Constructor
        Tiempo( int = 0, int = 0, int = 0); // constructor predeterminado
   // funciones establecer
   void establecerDatos( int, int, int ); // establece hora, minuto, segundo
   void estableceHora( int );   // establece hora
   void estableceMinutos( int );  // establece minuto
   void estableceSegundos( int ); // establece segundo
   // funciones obtener
   int obtieneHora();         // obtiene hora
   int obtieneMinutos();        // obtiene minuto
   int obtieneSegundos();       // obtiene segundo
   // función para capturar datos en pantalla
   void capturarDatos();
   // funciones para imprimir datos
```

```
   void imprimeUniversal();         // imprime la hora en formato universal
   void imprimeEstandar();          // imprime la hora en formato estándar
private:
   int hora;     // 0 - 23 (formato de reloj de 24 horas)
   int minutos;   // 0 - 59
   int segundos;  // 0 - 59
}; // fin de la clase Tiempo
#endif
```

tiempoSetGet.cpp

```
// Fig. 6.19: tiempoSetGet.cpp
// Definición de las funciones miembro de la clase Tiempo.
#include <iostream>
using std::cout;
using std::cin;
using std::endl;
#include <iomanip>
using std::setfill;
using std::setw;
// incluye la definición de la clase Tiempo desde tiempo3.h
#include "tiempoSetGet.h"

// CONSTRUCTOR
// función constructor para inicializar datos privados;
// llama a la función miembro estableceHora para asignar los variables;
// los valores predeterminados son 0 (vea la definicón de la clase)
Tiempo::Tiempo( int hr, int min, int seg )
{
       establecerDatos( hr, min, seg );
} // fin del constructor Tiempo

// FUNCIONES ESTABLECER
// establece los valores de hora, minuto y segundo
void Tiempo::establecerDatos( int h, int m, int s )
{
       estableceHora( h );
       estableceMinutos( m );
       estableceSegundos( s );
} // fin de la función establecerDatos

// establece el valor de hora
void Tiempo::estableceHora( int h )
{
   hora = ( h >= 0 && h < 24 ) ? h : 0;
} // fin de la función estableceHora

// establece el valor de minuto
void Tiempo::estableceMinutos( int m )
{
   minutos = ( m >= 0 && m < 60 ) ? m : 0;
} // fin de la función estableceMinutos

// establece el valor de segundo
void Tiempo::estableceSegundos( int s )
{
   segundos = ( s >= 0 && s < 60 ) ? s : 0;
} // fin de la función estableceSegundos

// FUNCIONES OBTENER
// devuelve el valor de hora
int Tiempo::obtieneHora()
{
   return hora;
} // fin de la función obtieneHora

// devuelve el valor de minuto
int Tiempo::obtieneMinutos()
{
   return minutos;
} // fin de la función obtieneMinutos

// devuelve el valor de segundo
int Tiempo::obtieneSegundos()
{
   return segundos;
} // fin de la función obtieneSegundo

// FUNCIÓN PARA CAPTURAR DATOS
// captura datos en pantalla y utiliza la función de establecer datos
void Tiempo::capturarDatos()
{
       int h, m, s = 0;
       cout << "Hora: ";
       cin >> h;
       cout << "Minutos: ";
       cin >> m;
       cout << "Segundos: ";
       cin >> s;
```

```
        establecerDatos(h, m, s);
}

// FUNCIONES IMPRIMIR
// imprime Tiempo en formato universal
void Tiempo::imprimeUniversal()
{
   cout << setfill( '0' ) << setw( 2 ) << hora << ":"
        << setw( 2 ) << minutos << ":"
        << setw( 2 ) << segundos;

} // fin de la función imprimeUniversal

// imprime Tiempo en formato estándar
void Tiempo::imprimeEstandar()
{
   cout << ( ( hora == 0 || hora == 12 ) ? 12 : hora % 12 )
        << ":" << setfill( '0' ) << setw( 2 ) << minutos
        << ":" << setw( 2 ) << segundos
        << ( hora < 12 ? " AM" : " PM" );
} // fin de la funcion imprimeEstandar
```

ejemploSetGet.cpp

```
// Fig. 6.20: fig06_20.cpp
// Demostración de las funciones establecer y obtener de la clase Tiempo
#include <iostream>
using std::cout;
using std::endl;
// incluye la definición de la clase Tiempo
#include "tiempoSetGet.h"

int main()
{
   Tiempo t;              // crea el objeto de Tiempo
   // Mostrando los datos establecidos con el constructor predeterminado
   cout << "Valor inicial de t: ";
   t.imprimeEstandar();

   // EJEMPLO DE USO FUNCIONES ESTABLECER
   // Modifica todos los datos con la función establecerDatos
   t.establecerDatos(8, 40, 10);
   // Mostrando los datos modificados
   cout << "\nDatos modificados con la función establecerDatos: ";
   t.imprimeEstandar();

   // Modifica la hora mediante las funciones establecer individuales
   t.estableceHora( 17 );     // establece hora en un valor válido
   t.estableceMinutos( 34 );   // establece minuto en un valor válido
   t.estableceSegundos( 25 );   // establece segundo en un valor válido
   // Mostrando los datos modificados
   cout << "\nDatos modificados con las funciones establecer individuales: ";
   t.imprimeEstandar();
   cout << endl;

   // Capturando los datos en pantalla
   cout << "Ingrese los datos para t: " << endl;
   t.capturarDatos();
   // Mostrando los datos modificados
   cout << "\nDatos modificados con la función capturarDatos: ";
   t.imprimeEstandar();

   // EJEMPLO DE USO FUNCIONES OBTENER
   // utilice funciones obtener (get) para mostrar los valores de hora, minuto y segundo
   cout << "\nValores individuales de cada dato miembro del objeto t:\n"
        << "  Hora: " << t.obtieneHora()
        << "  Minuto: " << t.obtieneMinutos()
        << "  Segundo: " << t.obtieneSegundos();

   // Sumándole una hora a un objeto de tipo tiempo, con ayuda de las funciones obtener
   int h = 0;
   h = t.obtieneHora();
   h++;
   t.estableceHora(h);
   // Mostrando los datos modificados
   cout << "\nDatos modificados después de sumarle 1 a la hora: ";
   t.imprimeEstandar();
   cout << endl;

   system ("Pause");
   return 0;
} // fin de main
```

Ejemplo de Clase Fecha Completa

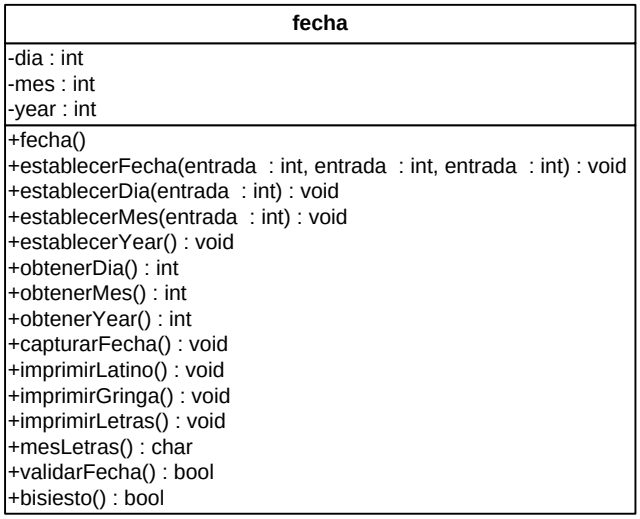
Planteamiento del Ejercicio

Sencillo. Hay que crear la clase fecha con todos los datos miembro y funciones que se especifican en el diagrama de clase, que muestra en esta página, y luego hay que elaborar un programa cliente que demuestre el uso de cada una de esas funciones.

Puntos relevantes mostrados en este ejemplo:

- Definición, declaración y uso de funciones set y get.
- Definición, declaración y uso de funciones booleanas.
- Funciones principales y funciones de utilidad.

Diagrama de Clase Fecha



Archivo de Encabezado

Fecha.h

```
// Clase Fecha
class fecha
{
private:
        int dia;
        int mes;
        int year;

public:
        // Constructor
        fecha(int = 1, int = 1, int = 1900);

        // Funciones principales
        void establecerFecha(int, int, int);
        void establecerDia(int);
        void establecerMes(int);
        void establecerYear(int);
        int obtenerDia();
        int obtenerMes();
        int obtenerYear();
        void capturarFecha();
        void imprimirLatino();
        void imprimirGringa();
        void imprimirLetras();

        // Funciones de utilidad

        char *mesLetras();
        bool validarFecha();
```

```
        bool bisiesto();
};
```

Archivo de funciones

Fecha.cpp

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

fecha::fecha(int d, int m, int y)
{       establecerFecha(d, m, y);
}

void fecha::establecerFecha(int d, int m, int y)
{       establecerDia(d);
        establecerMes(m);
        establecerYear(y);
        if (validarFecha() == false)
        {
                cout << "Fecha invalida, se establecera la fecha en 01/01/1900" << endl;
                establecerFecha(1, 1, 1900);
        }
}

void fecha::establecerDia(int d)
{       if (d < 1 || d > 31)
        {       cout << "Dia invalido, se establecera el día en 1" << endl;
                dia = 1;
        }
        else
                dia = d;
}

void fecha::establecerMes(int m)
{
        if (m < 1 || m > 12)
        {       cout << "Mes invalido, se establecera el mes en 1" << endl;
                mes = 1;
        }
        else
                mes = m;
}

void fecha::establecerYear(int y)
{
        if (y < 0 || y > 9999)
        {       cout << "Año invalido, se establecera el año en 1900" << endl;
                year = 1900;
        }
        else
                year = y;
}

bool fecha::validarFecha()
{       bool resultado = true; // True = Fecha correcta
        switch (mes)
        {
        case 4:
                if (dia > 30)
                        resultado = false;
                break;
        case 6:
                if (dia > 30)
                        resultado = false;
                break;
        case 9:
                if (dia > 30)
                        resultado = false;
                break;
        case 11:
                if (dia > 30)
                        resultado = false;
                break;
        case 2:
                if (dia > 28 && bisiesto() == false)
                        resultado = false;
                else
                        if (dia > 29)
                                resultado = false;
                        break;
        default:
                resultado = true;
        }
        return resultado;
}
```

```
bool fecha::bisiesto()
{       bool bis;
        int residuo = 0;

        residuo = year % 4;
        if (residuo > 0)
                bis = false;
        else
                bis = true;

        return bis;
}

int fecha::obtenerDia()
{       return dia;
}

int fecha::obtenerMes()
{       return mes;
}

int fecha::obtenerYear()
{       return year;
}

void fecha::capturarFecha()
{       int d, m, y;

        cout << "Dia: ";
        cin >> d;
        cout << "Mes: ";
        cin >> m;
        cout << "Año: ";
        cin >> y;
        establecerFecha(d, m, y);
}

void fecha::imprimirGringa()
{       cout << mes << "/" << dia << "/" << year << endl;
}

void fecha::imprimirLatino()
{       cout << dia << "/" << mes << "/" << year << endl;
}

void fecha::imprimirLetras()
{       cout << dia << " de " << mesLetras() << " de " << year << endl;
}

char *fecha::mesLetras()
{       char *m = " ";
        switch(mes)
        {
        case 1:
                m = "Enero";
                break;
        case 2:
                m = "Febrero";
                break;
        case 3:
                m = "Marzo";
                break;
        case 4:
                m = "Abril";
                break;
        case 5:
                m = "Mayo";
                break;
        case 6:
                m = "Junio";
                break;
        case 7:
                m = "Julio";
                break;
        case 8:
                m = "Agosto";
                break;
        case 9:
                m = "Septiembre";
                break;
        case 10:
                m = "Octubre";
                break;
        case 11:
                m = "Noviembre";
                break;
        case 12:
                m = "Diciembre";
```

```
                break;
        }

        return m;
}
```

Programa Cliente

testingFecha.cpp

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

int main()
{       fecha f;
        // Capturando datos
        f.capturarFecha();

        // Determinando si la fecha es valida
        if (f.validarFecha() == false)
        {       cout << "Fecha no valida" << endl;
        }
        else
        {       cout << "Fecha OK" << endl;
        }

        // Determinando si el year es bisiesto
        if (f.bisiesto())
        {       cout << "Es bisiesto" << endl;
        }
        else
                cout << "No es bisiesto" << endl;

        // Imprimiendo datos
        f.imprimirGringa();
        f.imprimirLatino();
        f.imprimirLetras();

        // Modificando los datos de la fecha de forma individual
        f.establecerDia(20);
        f.establecerMes(12);
        f.establecerYear(1999);

        // Imprimiendo los datos nuevamente
        cout << "Fecha con valores modificados individualmente: " << endl;
        f.imprimirGringa();
        f.imprimirLatino();
        f.imprimirLetras();

        // Modificando todos los datos de un solo
        f.establecerFecha(27, 11, 2001);

        //Imprimiendo nuevos datos, solo en letras
        cout << "Fecha con valores modificados todos de una vez: " << endl;
        f.imprimirGringa();
        f.imprimirLatino();
        f.imprimirLetras();

        // Mostrando los datos individualmente
        cout << "\nValores actuales de los datos miembro de objeto f: " << endl;
        cout << "dia: " << f.obtenerDia();
        cout << "\nmes: " << f.obtenerMes();
        cout << "\nmes en letras: " << f.mesLetras();
        cout << "\nyear: " << f.obtenerYear() << endl;
        system("pause");
}
```

Salida del Programa

```
        Dia: 31
        Mes: 12
        A±o: 2006
        Fecha OK
        No es bisiesto
        12/31/2006
        31/12/2006
        31 de Diciembre de 2006
        Fecha con valores modificados individualmente:
        12/20/1999
        20/12/1999
        20 de Diciembre de 1999
        Fecha con valores modificados todos de una vez:
        11/27/2001
        27/11/2001
        27 de Noviembre de 2001
        Valores actuales de los datos miembro de objeto f:
        dia: 27
```

```
mes: 11
mes en letras: Noviembre
year: 2001
Press any key to continue . . .
```

Composición

Ejemplo de Composición con Clase Fecha

En este ejemplo se demuestra uno de los conceptos fundamentales de la Programación Orientada a Objetos: La Composición.

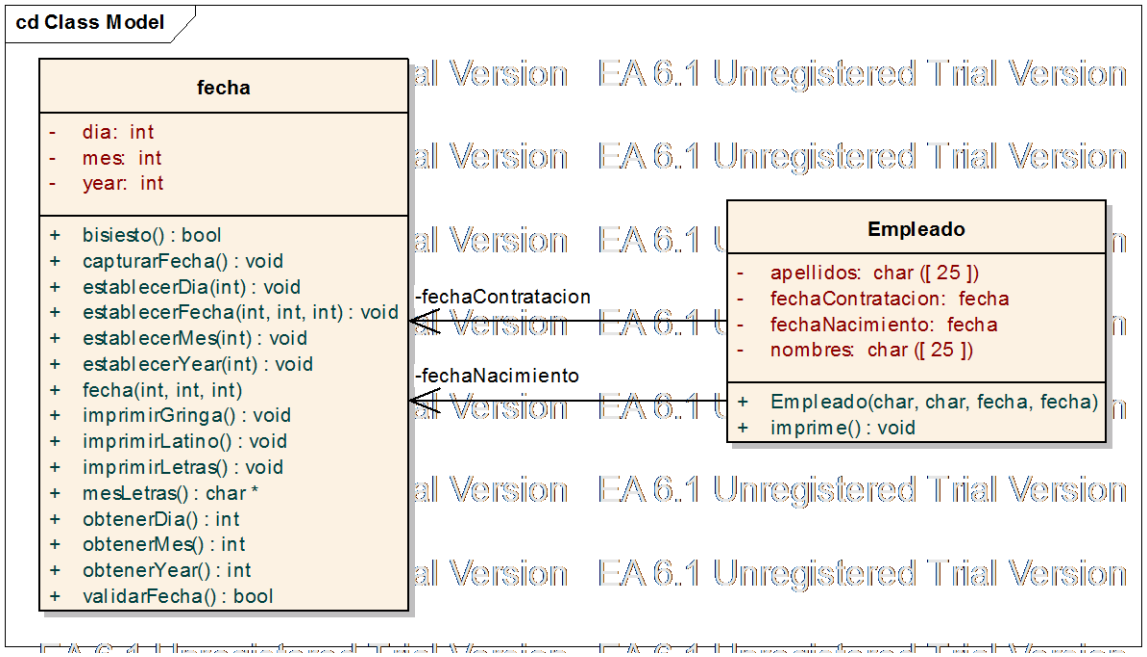
Mediante la composición a una clase se le pueden crear miembros (datos o funciones) cuyo tipo sea una clase previamente creada que sea incorporada al proyecto.

Por ejemplo, si ya hemos creado una clase fecha que validar datos, captura, e imprime fechas en diferentes formatos, es lógico pensar que dicha clase puede utilizarse fechas de nacimiento de empleados, en una clase pensada para definir empleados ¿no?

Recuerde que definir una clase es equivalente a definir un nuevo tipo de datos del lenguaje.

En este ejemplo se define una clase empleado, la cual, además de manejar nombres y apellidos de empleados, como datos miembro, utiliza la clase fecha como un tipo de datos para declarar la fecha de nacimiento y la fecha de contratación del empleado

Diagrama de clases



Código fuente

FECHA.H

```
// Definición de la clase Fecha.
// Funciones miembro definidas en fecha.cpp.
#ifndef fecha_H
#define fecha_H
// Clase Fecha
class fecha
{private:
        int dia;
        int mes;
        int year;
public:
        // Constructor
        fecha(int = 1, int = 1, int = 1900);

        // Funciones principales
        void establecerFecha(int, int, int);
        void establecerDia(int);
        void establecerMes(int);
        void establecerYear(int);
        int obtenerDia();
```

```
        int obtenerMes();
        int obtenerYear();
        void capturarFecha();
        void imprimirLatino();
        void imprimirGringa();
        void imprimirLetras();

        // Funciones de utilidad
        char *mesLetras();
        bool validarFecha();
        bool bisiesto();
};
#endif
```

FECHA.CPP

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

fecha::fecha(int d, int m, int y)
{       establecerFecha(d, m, y);
}

void fecha::establecerFecha(int d, int m, int y)
{       establecerDia(d);
        establecerMes(m);
        establecerYear(y);
        if (validarFecha() == false)
        {       cout << "Fecha invalida, se establecera la fecha en 01/01/1900" << endl;
                establecerFecha(1, 1, 1900);
        }
}

void fecha::establecerDia(int d)
{       if (d < 1 || d > 31)
        {       cout << "Dia invalido, se establecera el día en 1" << endl;
                dia = 1;
        }
        else
                dia = d;
}

void fecha::establecerMes(int m)
{       if (m < 1 || m > 12)
        {       cout << "Mes invalido, se establecera el mes en 1" << endl;
                mes = 1;
        }
        else
                mes = m;
}

void fecha::establecerYear(int y)
{       if (y < 0 || y > 9999)
        {       cout << "Año invalido, se establecera el año en 1900" << endl;
                year = 1900;
        }
        else
                year = y;
}

bool fecha::validarFecha()
{       bool resultado = true; // True = Fecha correcta
        switch (mes)
        {
        case 4:
                if (dia > 30)
                        resultado = false;
                break;
        case 6:
                if (dia > 30)
                        resultado = false;
                break;
        case 9:
                if (dia > 30)
                        resultado = false;
                break;
        case 11:
                if (dia > 30)
                        resultado = false;
                break;
        case 2:
                if (dia > 28 && bisiesto() == false)
                        resultado = false;
                else
                        if (dia > 29)
                                resultado = false;
                        break;
```

```
        default:
                resultado = true;
        }
        return resultado;
}

bool fecha::bisiesto()
{       bool bis;
        int residuo = 0;

        residuo = year % 4;
        if (residuo > 0)
                bis = false;
        else
                bis = true;

        return bis;
}

int fecha::obtenerDia()
{       return dia;
}

int fecha::obtenerMes()
{       return mes;
}

int fecha::obtenerYear()
{       return year;
}

void fecha::capturarFecha()
{       int d, m, y;

        cout << "Dia: ";
        cin >> d;
        cout << "Mes: ";
        cin >> m;
        cout << "Año: ";
        cin >> y;
        establecerFecha(d, m, y);
}

void fecha::imprimirGringa()
{       cout << mes << "/" << dia << "/" << year << endl;
}

void fecha::imprimirLatino()
{       cout << dia << "/" << mes << "/" << year << endl;
}

void fecha::imprimirLetras()
{       cout << dia << " de " << mesLetras() << " de " << year << endl;
}

char *fecha::mesLetras()
{       char *m = " ";
        switch(mes)
        {
        case 1:
                m = "Enero";
                break;
        case 2:
                m = "Febrero";
                break;
        case 3:
                m = "Marzo";
                break;
        case 4:
                m = "Abril";
                break;
        case 5:
                m = "Mayo";
                break;
        case 6:
                m = "Junio";
                break;
        case 7:
                m = "Julio";
                break;
        case 8:
                m = "Agosto";
                break;
        case 9:
                m = "Septiembre";
                break;
        case 10:
                m = "Octubre";
```

```
            break;
        case 11:
            m = "Noviembre";
            break;
        case 12:
            m = "Diciembre";
            break;
        }
        return m;
};
```

EMPLEADO.H

```
// Definición de la clase Empleado.
// Funciones miembro definidas en empleado1.cpp.
#ifndef EMPL_H
#define EMPL_H
// incluye la definición de la clase Fecha de fecha.h
#include "fecha.h"

class Empleado
{
public:
   Empleado( char [25], char [25], fecha, fecha);
   void imprime();
private:
   char nombres[ 25 ];
   char apellidos[ 25 ];
   fecha fechaNacimiento;   // composición: objeto miembro
   fecha fechaContratacion; // composición: objeto miembro
}; // fin de la clase Empleado
#endif
```

EMPLEADO.CPP

```
// Definición de las funciones miembro para la clase Empleado.
#include <iostream>
using std::cout;
using std::endl;
#include "empleado.h"  // Definición de la clase Empleado
#include "fecha.h"     // Definición de la clase Fecha

// Este constructor requiere recibir todos los datos para poder inicializar un objeto de tipo empleado
Empleado::Empleado( char n [25], char a [25], fecha fNac, fecha fCont)
{
   // copia primero en nombres y se asegura de que cabe
   strcpy_s(nombres, n);
   strcpy_s(apellidos, a);
   fechaNacimiento = fNac;
   fechaContratacion = fCont;

   // muestra el objeto Empleado para determinar cuándo se llama al constructor
   cout << "Constructor del objeto Empleado: "
        << nombres << ' ' << apellidos << endl;
} // fin del constructor Empleado

// imprime el objeto Empleado
void Empleado::imprime()
{
   cout << apellidos << ", " << nombres << "\nContratado el: ";
   fechaContratacion.imprimirLetras();
   cout << "  Fecha de nacimiento: ";
   fechaNacimiento.imprimirLetras();
   cout << endl;
} // fin de la función imprime
```

EJEMPLO_COMPOSICION.CPP

```
// Demostración de composición --un objeto con objetos miembro.
#include <iostream>
using std::cout;
using std::endl;
#include "empleado.h"  // Definición de la clase Empleado

int main()
{
   /*fecha nacimiento( 17, 24, 1949 );
   fecha contratacion( 3, 12, 1988 );*/
        fecha nacimiento1;
        fecha nacimiento2;
        fecha contratacion;

        cout << "Fecha de Nacimiento para primer empleado: " << endl;
        nacimiento1.capturarFecha();
        cout << "Fecha de Nacimiento para segundo empleado: " << endl;
        nacimiento2.capturarFecha();
        cout << "Fecha de Contratación para ambos empleados: " << endl;
        contratacion.capturarFecha();
```

```
   Empleado gerente( "Juan", "Perez", nacimiento1, contratacion );
   Empleado conserje("Pedro", "Lopez", nacimiento2, contratacion);

   cout << '\n';
   gerente.imprime();
   conserje.imprime();

   system("pause");
   return 0;
} // fin de main
```

SALIDA DEL PROGRAMA

```
Fecha de Nacimiento para primer empleado:
Dia: 31
Mes: 12
A±o: 1978
Fecha de Nacimiento para segundo empleado:
Dia: 25
Mes: 02
A±o: 1980
Fecha de Contrataci¾n para ambos empleados:
Dia: 01
Mes: 07
A±o: 1999
Constructor del objeto Empleado: Juan Perez
Constructor del objeto Empleado: Pedro Lopez

Perez, Juan
Contratado el: 1 de Julio de 1999
  Fecha de nacimiento: 31 de Diciembre de 1978

Lopez, Pedro
Contratado el: 1 de Julio de 1999
  Fecha de nacimiento: 25 de Febrero de 1980

Presione una tecla para continuar . . .
```

Herencia

Primer Ejemplo de Herencia: Clases Persona y Estudiante

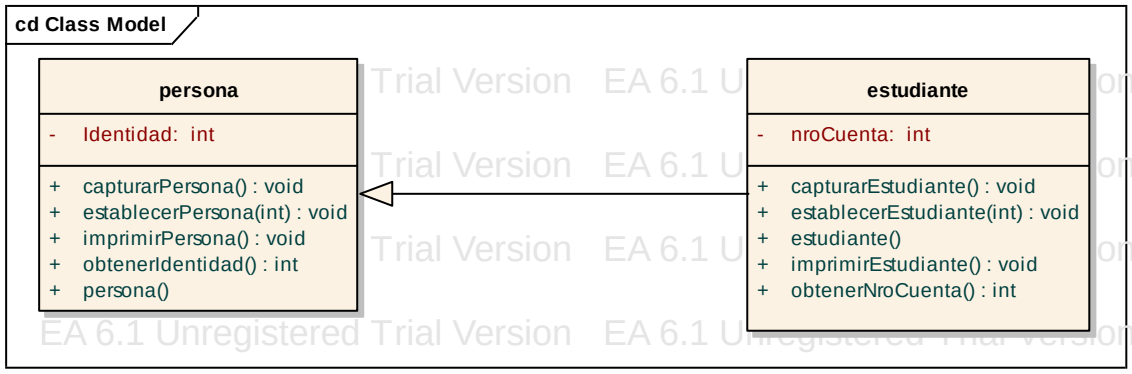
La Herencia es la cualidad más importante de la Programación Orientada a Objetos. Es lo que dará mayor potencia y productividad, ahorrando horas de codificación y de depuración de errores.

La herencia permite reutilizar e las subclases todo el código escrito para las superclases, codificando únicamente las diferencias que entre las superclases y las subclases.

A la clase heredada se le llama subclase o clase hija y a la clase que hereda se le llama superclase o clase padre.

En el siguiente ejemplo, se puede observar cómo la clase empleado **reutiliza** el código definido para la clase persona y solo amplía lo que es necesario para manejar un tipo *especializado* de persona: la clase empleado

Diagrama de clases



Código fuente

ejemploHerencia.h

```
#ifndef ejemplo_h
#define ejemplo_h

class persona
{
private:
	int Identidad;
public:
	persona();
	void capturarPersona();
	void establecerPersona(int);
	int obtenerIdentidad();
	void imprimirPersona();
};
class estudiante : public persona
{
private:
	int nroCuenta;
public:
	estudiante();
	void capturarEstudiante();
	void establecerEstudiante(int);
	int obtenerNroCuenta();
	void imprimirEstudiante();
};
#endif
```

ejemploHerencia.cpp

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

```
// Funciones de Clase persona
persona::persona()
{
	int p = 0;
	establecerPersona(p);
}
void persona::establecerPersona(int p)
{
	Identidad = p;
}
int persona::obtenerIdentidad()
{
	return Identidad;
}
void persona::capturarPersona()
{
	int id = 0;
	cout << "Ingrese Identidad: ";
	cin >> id;
	establecerPersona(id);
}
void persona::imprimirPersona()
{
	cout << "\nIdentidad de Persona: " << Identidad;
}

// Funciones de la clase estudiante
estudiante::estudiante()
{
	int cuenta = 0;
	establecerEstudiante(cuenta);
}
void estudiante::establecerEstudiante(int c)
{
	nroCuenta = c;
}
int estudiante::obtenerNroCuenta()
{
	return nroCuenta;
}
void estudiante::capturarEstudiante()
{
	int nroC = 0;
	cout << "Ingrese número de cuenta: ";
	cin >> nroC;
	establecerEstudiante(nroC);
}
void estudiante::imprimirEstudiante()
{
	cout << "\nNro Cuenta Estudiante: " << nroCuenta;
}
```

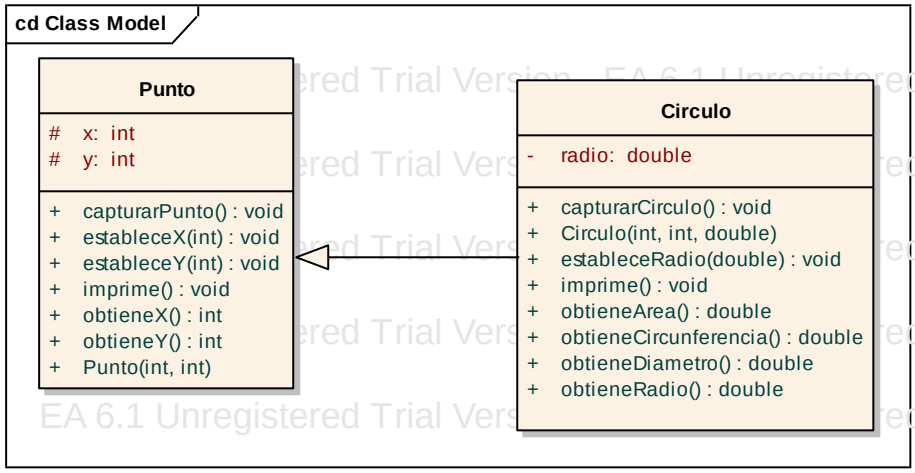
testingEstudiante.cpp

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

int main()
{
	estudiante e1;
	cout << "\nCapturando los datos de un estudiante: " << endl;
	e1.capturarPersona();
	e1.capturarEstudiante();
	cout << "\nImprimiendo los datos de un estudiante: " << endl;
	e1.imprimirPersona();
	e1.imprimirEstudiante();
	cout << endl;
	system("pause");
	return 0;
}
```

Segundo Ejemplo de Herencia: Clases Punto y Círculo

En este ejemplo se define la clase Punto como clase base y a partir de ella se define una subclase llamada Circulo, la cual hereda los datos de punto. Un punto en el círculo representaría el centro del mismo.
Luego para la subclase Circulo se define un único dato miembro: el radio.
Radio es el único dato miembro que se define, ya que los demás datos que describen un círculo (diámetro, circunferencia y área) pueden ser calculados en base al radio, y por tanto estarían más correctamente definidos como funciones, que como datos miembro.

Diagrama de clases



Código fuente

Punto.h

```
// La definición de la clase Punto representa un par de coordeandas x-y.
#ifndef PUNTO_H
#define PUNTO_H
class Punto {
public:
   Punto( int = 0, int = 0 ); // constructor predeterminado
   void estableceX( int );  // establece x en un par de coordenadas
   int obtieneX() const;    // devuelve x desde el par de coordenadas
   void estableceY( int );  // establece y en un par de coordenadas
   int obtieneY() const;    // devuelve y desde un par de coordenadas
   void imprime() const;  // despliega el objeto Punto
   void capturarPunto(); // para capturar datos en pantalla
protected:
   int x;  // parte x del par de coordenadas
   int y;  // parte y del par de coordenadas
}; // fin de la clase Punto
#endif
```

Punto.cpp

```
// Definición de las funciones miembro para la clase Punto.
#include <iostream>
using namespace std;
#include "punto.h"   // definición de la clase Punto

// constructor predeterminado
Punto::Punto( int valorX, int valorY )
{
   x = valorX;
   y = valorY;
} // fin del constructor Punto2

// establece x en el par de coordenadas
void Punto::estableceX( int valorX )
{
   x = valorX; // no requiere validación
} // fin de la función estableceX

// devuelve x desde el par de coordenadas
int Punto::obtieneX() const
```

```
{   return x;
} // fin de la función obtieneX

// establece y en un par de coordenadas
void Punto::estableceY( int valorY )
{
   y = valorY; // no requiere validación
} // fin de la función estableceY

// devuelve y desde el par de coordenadas
int Punto::obtieneY() const
{   return y;
} // fin de la función obtieneY

// muestra el objeto Punto2
void Punto::imprime() const
{
   cout << '[' << x << ", " << y << ']';
} // fin de la función imprime

void Punto::capturarPunto()
{
      int valorX = 0;
      int valorY = 0;

      cout << "Valor de X: ";
      cin >> valorX;
      cout << "Valor de Y: ";
      cin >> valorY;

      estableceX(valorX);
      estableceY(valorY);
}
```

Circulo.h

```
// La clase Circulo contiene un par de coordenadas x-y y un radio.
#ifndef CIRCULO_H
#define CIRCULO_H
#include "punto.h"  // definición de la clase Punto

class Circulo : public Punto {
public:
   // constructor predeterminado
   Circulo( int = 0, int = 0, double = 0.0 );
   void estableceRadio( double );   // establece el radio
   double obtieneRadio() const;   // devuelve el radio
   double obtieneDiametro() const;   // devuelve el diámetro
   double obtieneCircunferencia() const; // devuelve la circunferencia
   double obtieneArea() const;           // devuelve el área
   void imprime() const;          // muestra el objeto Circulo
   void capturarCirculo(); // captura datos de un círculo
private:
   double radio;  // radio de Circulo
}; // fin de la clase Circulo
#endif
```

Circulo.cpp

```
// Definición de las funciones miembro de la clase Circulo.
#include <iostream>
using namespace std;
#include "circulo.h"   // definición de la clase Circulo

// constructor predeterminado
Circulo::Circulo( int valorX, int valorY, double valorRadio )
{
   x = valorX;
   y = valorY;
   estableceRadio( valorRadio );
} // fin del constructor Circulo

// establece radio
void Circulo::estableceRadio( double valorRadio )
{
   radio = ( valorRadio < 0.0 ? 0.0 : valorRadio );
} // fin de la función estableceRadio

// devuelve el radio
```

```
double Circulo::obtieneRadio() const
{   return radio;
} // fin de la función obtieneRadio

// calcula y devuelve el diámetro
double Circulo::obtieneDiametro() const
{   return 2 * radio;
} // fin de la función obtieneDiametro

// calcula y devuelve la circunferencia
double Circulo::obtieneCircunferencia() const
{   return 3.14159 * obtieneDiametro();
} // fin de la función obtieneCircunferencia

// calcula y devuelve el área
double Circulo::obtieneArea() const
{
   return 3.14159 * radio * radio;
} // fin de la función obtieneArea

// muestra el objeto Circulo2
void Circulo::imprime() const
{
   cout << "Centro = [" << x << ", " << y << ']' << "; Radio = " << radio;
} // fin de la función imprime

void Circulo::capturarCirculo()
{
       double r = 0;

       capturarPunto();
       cout << "\nRadio: ";
       cin >> r;

       estableceRadio(r);
}
```

ejemploHerenciaCirculo.cpp

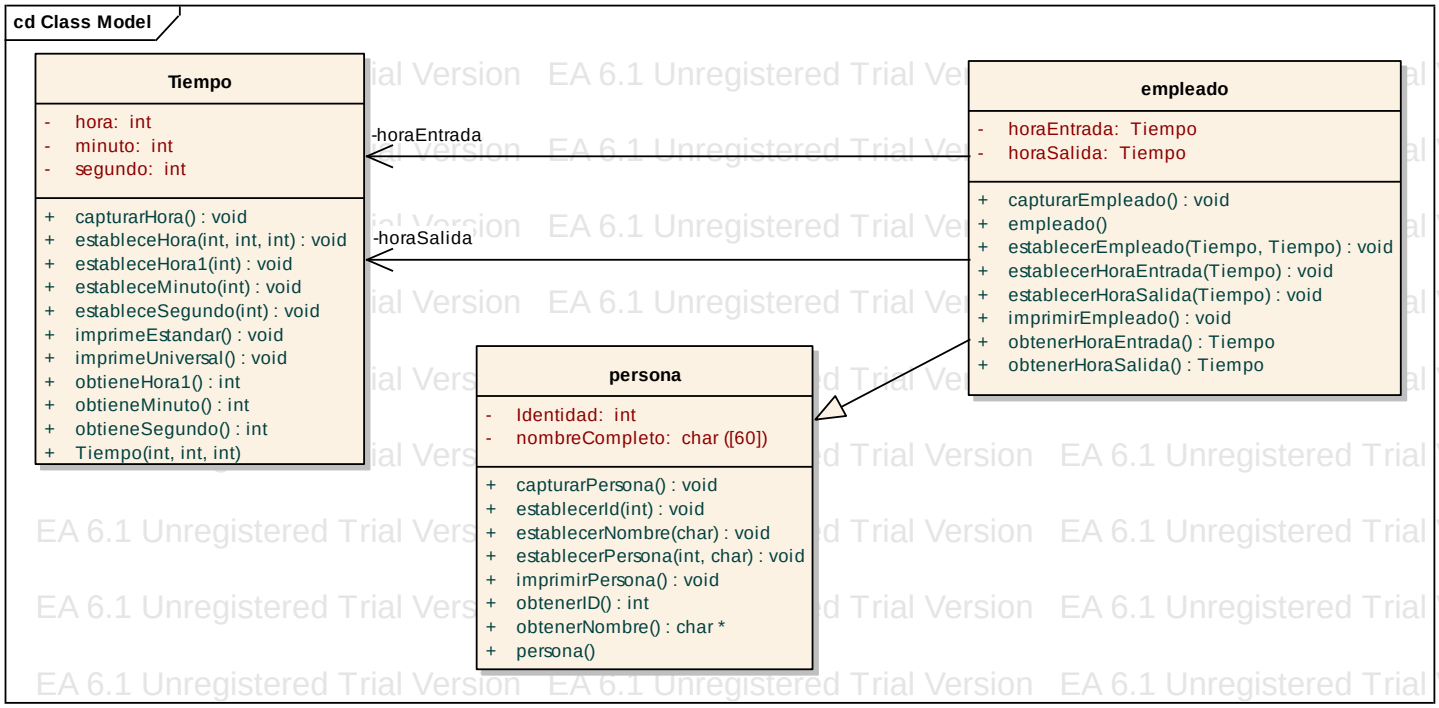
```
#include <iostream>
using namespace std;
#include "circulo.h"
#include "punto.h"
int main()
{
       Circulo c1;
       c1.capturarCirculo();
       c1.imprime();
       cout << "\nEl diametro es: " << c1.obtieneDiametro();
       cout << "\nLa circunferencia es: " << c1.obtieneCircunferencia() <<endl;
       cout << "\nEl area es: " << c1.obtieneArea() << endl;
       system("pause");
       return 0;
}
```

Ejemplo Combinado de Herencia y Composición

En este ejemplo se aplican los conceptos de herencia y composición, para mostrar una panorámica completa de lo que representa la reutilización de código y el encapsulamiento en la Programación Orientada a Objetos.

Como podrá verse la clase empleado está conformada en su totalidad por código reutilizado, tanto de la clase persona, de la que hereda los datos descriptivos de una persona (Identidad y Nombre), como de la clase Tiempo la cual utiliza para definir datos descriptivos del empleado como ser la Hora de Entrada y la Hora de Salida.

Diagrama de Clases



Código fuente

ejemploHerencia.h

```
#ifndef herencia_h
#define herencia_h
// Clase Tiempo
class Tiempo {
public:
   // constructor predeterminado
        Tiempo( int = 0, int = 0, int = 0);
   // funciones establecer
   void estableceHora( int, int, int );  // establece hora
   void estableceHora1( int );           // establece hora
   void estableceMinuto( int );          // establece minuto
   void estableceSegundo( int );         // establece segundo
   // funciones obtener (por lo general se declaran const)
   int obtieneHora1() const;             // devuelve hora
   int obtieneMinuto() const;            // devuelve minuto
   int obtieneSegundo() const;           // devuelve segundo
   // funciones de impresión (por lo general se declaran const)
   void imprimeUniversal() const;    // imprime la hora en formato universal
   void imprimeEstandar();           // imprime la hora en formato estándar
   // funcion para capturar los datos de una hora
   void capturarHora();
private:
   int hora;    // 0 - 23 (formato de reloj de 24 horas)
   int minuto;  // 0 - 59
   int segundo; // 0 - 59
}; // fin de la clase Tiempo

// Clase Persona
class persona
{
private:
        int Identidad;
        char nombreCompleto [60];
public:
```

```
        // El constructor no esta preparado para recibir argumentos, por lo tanto no pueden establecerse
        // valores iniciales a los datos, al momento de declarar un objeto
        persona();
        // Funciones Establecer
        void establecerPersona (int, char [60] );
        void establecerId(int);
        void establecerNombre(char [60]);
        // Funciones Obtener
        int obtenerID();
        char *obtenerNombre();
        // Funcion para capturar los datos de personas
        void capturarPersona();
        // Funcion para visualizar los datos de una persona
        void imprimirPersona();
};

// En esta clase empleado puede observarse un ejemplo tanto de herencia (declarando que empleado hereda de
persona)
// y de composicion, declarando la hora de entrada y la hora de salida de los empleados de tipo Tiempo
class empleado : public persona
{
public:
        // Constructor, no recibe parametros
        empleado();
        // Funciones Establecer
        void establecerEmpleado(Tiempo, Tiempo);
        void establecerHoraEntrada( Tiempo );
        void establecerHoraSalida(  Tiempo );
        // Funciones Obtener
        Tiempo obtenerHoraEntrada();
        Tiempo obtenerHoraSalida();
        // Funcion para captura de datos
        void capturarEmpleado();
        // Funcion para imprimir los datos de un empleado
        void imprimirEmpleado();
private:
        // Datos miembro de la clase empleado. Notese que no se incluyen ID y Nombre, debido a que
        // esos datos son HEREDADOS de la clase persona
        Tiempo horaEntrada;
        Tiempo horaSalida;
};
#endif
```

ejemploHerencia.cpp

```
// DEFINICION DE LAS FUNCIONES MIEMBRO DE LA CLASE Tiempo.
#include <iostream>
using std::cout;
using std::cin;
using std::endl;
#include <iomanip>
using std::setfill;
using std::setw;
// incluye la definición de la clase tiempo Tiempo desde ejemploHerencia.h
#include "ejemploHerencia.h"

// función constructor para inicializar datos privados;
// invoca a la función miembro estableceHora para establecer las variables;
// los valores predeterminados son 0 (vea la definición de la clase)
Tiempo::Tiempo( int hora, int minuto, int segundo )
{
   estableceHora( hora, minuto, segundo );

} // fin del constructor Tiempo

// establece los valores hora, minuto y segundo
void Tiempo::estableceHora( int hora, int minuto, int segundo )
{
   estableceHora1( hora );
   estableceMinuto( minuto );
   estableceSegundo( segundo );
} // fin de la función estableceHora

// establece el valor de hora
void Tiempo::estableceHora1( int h )
{
   hora = ( h >= 0 && h < 24 ) ? h : 0;
} // fin de la función estableceHora1

// establece el valor del minuto
void Tiempo::estableceMinuto( int m )
{
   minuto = ( m >= 0 && m < 60 ) ? m : 0;
} // fin de la función estableceMinuto

// establece el valor del segundo
void Tiempo::estableceSegundo( int s )
```

```
{
   segundo = ( s >= 0 && s < 60 ) ? s : 0;
} // fin de la función end function estableceSegundo

// devuelve el valor de hora
int Tiempo::obtieneHora1() const
{
   return hora;
} // fin de la función obtieneHora
// devuelve el valor del minuto
int Tiempo::obtieneMinuto() const
{
   return minuto;
} // fin de la función obtieneMinuto

// devuelve el valor del segundo
int Tiempo::obtieneSegundo() const
{
   return segundo;
} // fin de la función obtieneSegundo

// imprime Tiempo en formato universal
void Tiempo::imprimeUniversal() const
{
   cout << setfill( '0' ) << setw( 2 ) << hora << ":"
        << setw( 2 ) << minuto << ":"  << setw( 2 ) << segundo;
} // fin de la función imprimeUniversal

// imprime Tiempo en formato estándar
void Tiempo::imprimeEstandar()
{
   cout << ( ( hora == 0 || hora == 12 ) ? 12 : hora % 12 )
        << ":" << setfill( '0' ) << setw( 2 ) << minuto << ":" << setw( 2 ) << segundo
        << ( hora < 12 ? " AM" : " PM" );
} // fin de la función imprimeEstandar
// Captura datos para objetos de tipo Tiempo
void Tiempo::capturarHora()
{
       int h = 0;
       int m = 0;
       int s = 0;
       cout << "Hora: ";
       cin >> h;
       cout << "Minutos: ";
       cin >> m;
       cout << "Segundos: ";
       cin >> s;
       estableceHora(h, m, s);
}
// Definición de las FUNCIONES MIEMBRO de la clase Persona.
persona::persona()
{
       int id = 0;
       char nom [60];
       strcpy_s(nom, "");
       establecerPersona(id, nom);
}

void persona::establecerPersona(int i, char n [60])
{
       establecerId(i);
       establecerNombre(n);
}
void persona::establecerId(int i)
{
       Identidad = i;
}
void persona::establecerNombre(char n[60])
{
       strcpy_s(nombreCompleto, n);
}
int persona::obtenerID()
{
       return Identidad;
}
char *persona::obtenerNombre()
{
       return nombreCompleto;
}
void persona::capturarPersona()
{
       int id = 0;
       char nm [60];
       cout << "Numero de identidad: ";
       cin >> id;
       cin.ignore();
       cout << "Nombre: ";
       cin.getline(nm, 60);
```

```
        establecerPersona(id, nm);
}
void persona::imprimirPersona()
{
        cout << Identidad << " " << nombreCompleto << endl;
}

// DEFINICION DE LAS FUNCIONES MIEMBRO DE LA CLASE empleado
// Constructor
empleado::empleado()
{
        Tiempo he;
        Tiempo hs;
        establecerEmpleado(he, hs);
}
// Funciones establecer
void empleado::establecerEmpleado(Tiempo e, Tiempo s)
{
        establecerHoraEntrada(e);
        establecerHoraSalida(s);
}
void empleado::establecerHoraEntrada(Tiempo he)
{
        horaEntrada = he;
}
void empleado::establecerHoraSalida(Tiempo hs)
{
        horaSalida = hs;
}
// Funciones Obtener
Tiempo empleado::obtenerHoraEntrada()
{
        return horaEntrada;
}
Tiempo empleado::obtenerHoraSalida()
{
        return horaSalida;
}
// Funcion para capturar los datos
void empleado::capturarEmpleado()
{
        Tiempo e, s;
        cout << "Hora de Entrada: " << endl;
        e.capturarHora();
        cout << "Hora de Salida: " << endl;
        s.capturarHora();
        establecerEmpleado(e, s);
}
// Funcion para imprimir los datos de un empleado
void empleado::imprimirEmpleado()
{
        imprimirPersona();
        cout << "Hora de entrada: ";
        horaEntrada.imprimeUniversal();
        cout << "\nHora de Salida: ";
        horaSalida.imprimeUniversal();
        cout << endl;
}
```

programaCliente.cpp

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

int main()
{
        empleado e;
        e.capturarPersona();
        e.capturarEmpleado();
        cout << "----------- Imprimiendo datos de Prueba ------------" << endl;
        e.imprimirEmpleado();

        system("pause");
        return 0;
}
```

Ejemplo Herencia-Composición: Control de Entradas y Salidas de Empleados

Planteamiento del Problema

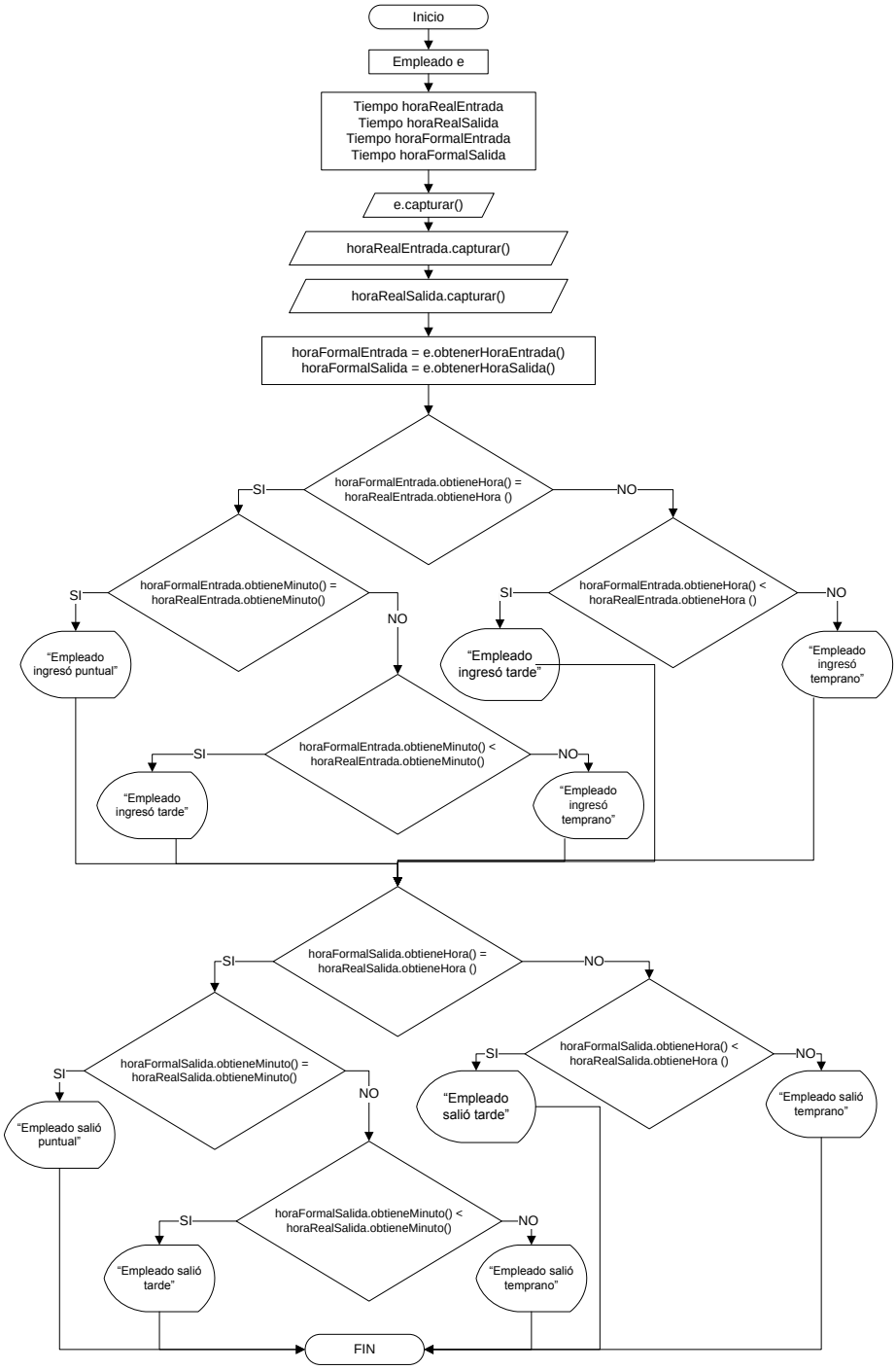
La Programación Orientada a Objetos no consiste simplemente en crear clases y programar funciones para las clases. Eso solo es la mitad del trabajo. El objetivo de la POO, al igual que cualquier estilo o metodología de programación, es el de resolver problemas reales.

Para ello se apoya en las clases que proporcionan estructuras y funciones en base sobre las cuales se definen objetos los cuales, mediante la invocación de funciones, ayudan a resolver el problema.

A continuación se muestra un ejemplo. En él se utilizan los mismos archivos encabezados y de funciones que en el ejemplo anterior, y se agrega un programa principal, el cual con objetos de las clases Tiempo, Persona y Empleado resuelve el siguiente requerimiento:

"Se necesita un programa para determinar si un empleado se presento a trabajar temprano, puntual o tarde. Además el programa debe determinar si el empleado salió de trabajar temprano, puntual o tarde".

Diagrama de Flujo



Código Fuente (solo del programa principal)

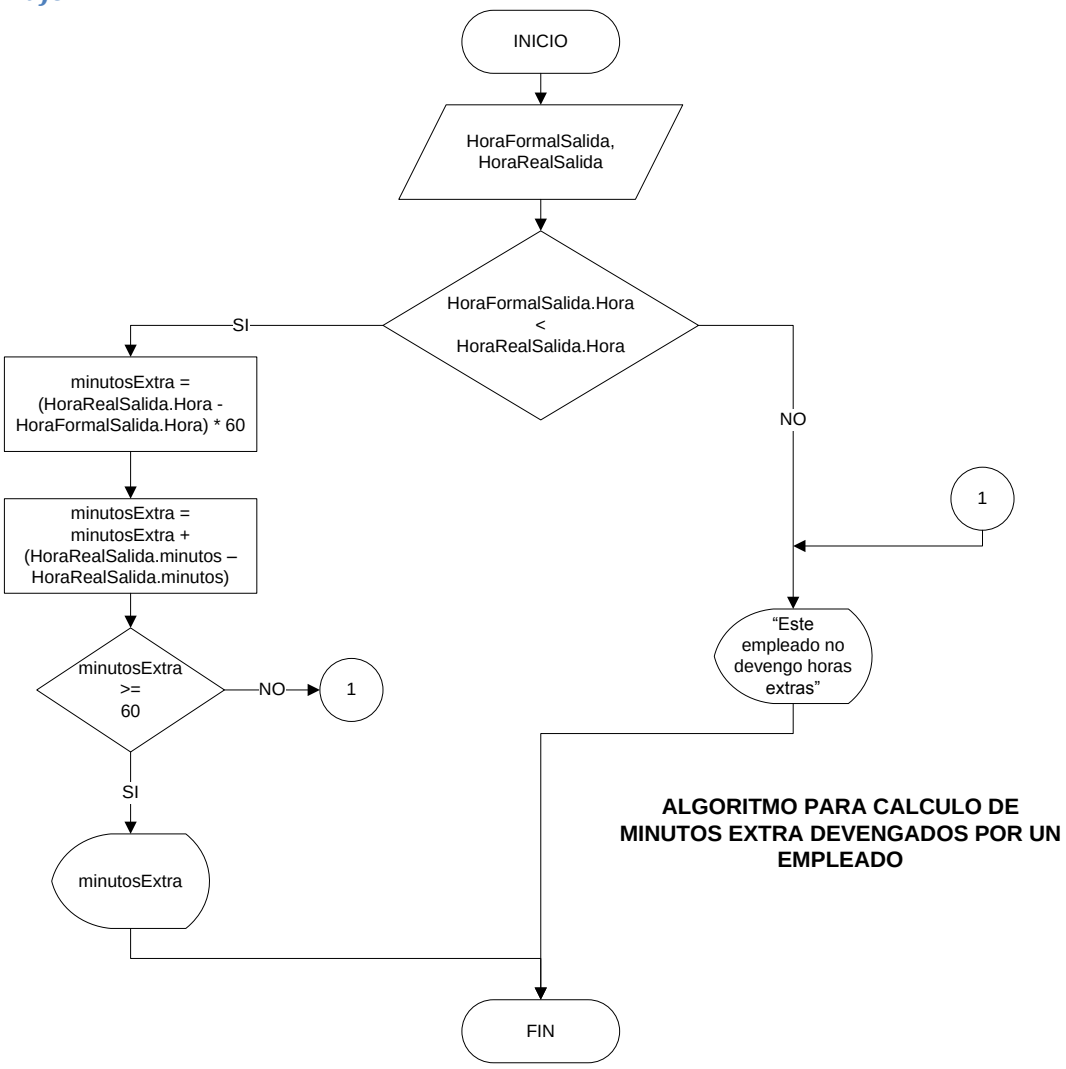
```
#include <iostream>
using namespace std;
#include "ejemploHerenciaComposicion.h"
int main()
{
	// Primer paso: Declarar un objeto de tipo empleado. Recuerde que aqui almacenara la ID, el nombre, la -
	//hora oficial de entrada, y la hora oficial de salida del empleado.
	empleado e;

	// Segundo Paso: Declarar los objetos en los que se almacenara las horas reales de entrada y salida del
	// empleado.
	Tiempo horaRealEntrada;
	Tiempo horaRealSalida;

	// Tercer paso: Capturar los datos del empleado
	cout << "Ingrese los datos del empleado: " << endl;
	e.capturarPersona();
	e.capturarEmpleado();

	// Cuarto paso: Capturar los datos de entrada y salida real del empleado
	cout << "\nIngrese hora real de entrada: " << endl;
	horaRealEntrada.capturarHora();
	cout << "\nIngrese hora real de salida: " << endl;
	horaRealSalida.capturarHora();

	// Quinto paso: Para comenzar a hacer las comparaciones es necesario extraer las horas de entrada y
	//salida del objeto empleado utilizando objetos de trabajo
	Tiempo horaFormalEntrada
	Tiempo horaFormalSalida;
	horaFormalEntrada = e.obtenerHoraEntrada();
	horaFormalSalida = e.obtenerHoraSalida();

	// Sexto paso: Determinar si el empleado ingreso puntual, tarde o temprano
	if (horaFormalEntrada.obtieneHora1() == horaRealEntrada.obtieneHora1())
	{
		if (horaFormalEntrada.obtieneMinuto() == horaRealEntrada.obtieneMinuto())
		{
			cout << "\nEl empleado ingreso puntual " << endl;
		}
		else
			if (horaFormalEntrada.obtieneMinuto() < horaRealEntrada.obtieneMinuto())
			{	cout << "\nEmpleado ingreso tarde " << endl;
			}
			else
			{	cout << "\nEmpleado ingreso temprano " << endl;
			}
	}
	else
	{
		if (horaFormalEntrada.obtieneHora1() < horaRealEntrada.obtieneHora1())
		{	cout << "\nEmpleado ingreso tarde " << endl;
		}
		else
		{	cout << "\nEmpleado ingreso temprano " << endl;
		}
	}

	// Septimo paso: Determinar si el empleado salio puntual, tarde o temprano
	if (horaFormalSalida.obtieneHora1() == horaRealSalida.obtieneHora1())
	{
		if (horaFormalSalida.obtieneMinuto() == horaRealSalida.obtieneMinuto())
		{	cout << "\nEl empleado salio puntual " << endl;
		}
		else
			if (horaFormalSalida.obtieneMinuto() < horaRealSalida.obtieneMinuto())
			{	cout << "\nEmpleado salio tarde " << endl;
			}
			else
			{	cout << "\nEmpleado salio temprano " << endl;
			}
	}
	else
	{
		if (horaFormalSalida.obtieneHora1() < horaRealSalida.obtieneHora1())
		{	cout << "\nEmpleado salio tarde " << endl;
		}
		else
		{	cout << "\nEmpleado salio temprano " << endl;
		}
	}
	system("pause");
	return 0;
}
```

Ejemplo Herencia-Composición: Programa para cálculo de tiempo extra trabajado por un empleado

Planteamiento del Problema

Se necesita un programa que sirva para determinar si debe o no pagarse horas extras a un empleado. Además debe determinar la cantidad de MINUTOS extras que deben pagársele.

Diagrama de Flujo



Código Fuente

```
#include <iostream>
using namespace std;
#include "ejemploHerenciaComposicion.h"
int main()
{

// Primer paso: Declarar un objeto de tipo empleado
	empleado e;

// Segundo Paso: Declarar los objetos en los que se almacenara la hora real de salida del empleado.
	Tiempo horaRealSalida;

// Tercer paso: Capturar los datos del empleado
	cout << "Ingrese los datos del empleado: " << endl;
	e.capturarPersona();
	e.capturarEmpleado();

// Cuarto paso: Capturar los datos de salida real del empleado
	cout << "\nIngrese hora real de salida: " << endl;
	horaRealSalida.capturarHora();

// Quinto paso: Para determinar las horas extras es necesario extraer la hora de
```

```
// salida del objeto empleado utilizando un objeto de trabajo
        Tiempo horaFormalSalida;
        horaFormalSalida = e.obtenerHoraSalida();

// Sexto paso: Determinar si el empleado trabajo tiempo adicional
        if (horaFormalSalida.obtieneHora1() < horaRealSalida.obtieneHora1())
        {
                int minutosExtra = (horaRealSalida.obtieneHora1() - horaFormalSalida.obtieneHora1()) * 60;
                minutosExtra += horaRealSalida.obtieneMinuto() - horaFormalSalida.obtieneMinuto();
                if (minutosExtra < 60)
                {
                        cout << "\nEmpleado no devengo horas extras" << endl;
                }
                else
                {
                        cout << "\Minutos extra trabajados: " << minutosExtra << endl;
                }
        }
        else
        {
                cout << "\nEmpleado no devengo horas extras" << endl;
        }
        system("pause");
        return 0;
}
```

Ejemplo Herencia-Composición: Electrónicos

Planteamiento del Problema

Se desea realizar una aplicación informática que sirva para capturar e imprimir los datos de facturas por la venta de aparatos electrónicos.
Como programador orientado a objetos Ud. ya habrá deducido que debe crear una clase factura.
Dicha clase debe contener los siguientes datos:

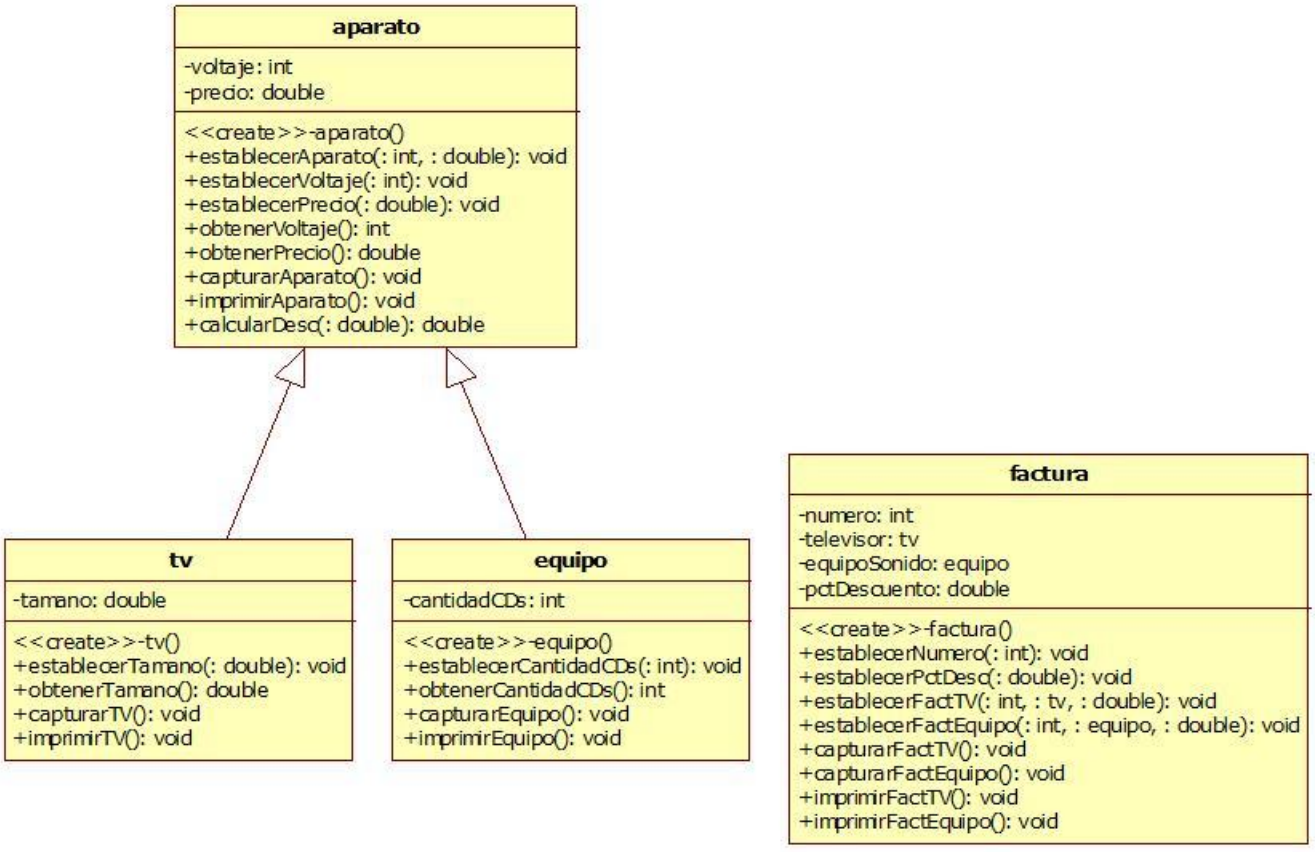
- Número de factura (int).
- Aparato que se vende. Se asume que cada factura contiene un solo aparato, sin embargo. ¡OJO! Ese aparato podría ser un TV o un equipo de sonido.
- Porcentaje de descuento (double).
- Opcional: Fecha de la factura, utilizando la clase fecha.

La clase factura debe contener las siguientes funciones:

- Constructor
- Función establecer (considerando las limitaciones de tiempo, no se requieren funciones establecer y obtener para cada dato miembro).
- Función para captura de datos (De nuevo…¡OJO! El aparato podría ser TV o equipo de sonido).
- Función para imprimir datos (De nuevo…bueno ya sabe).

Nota: Se recomienda incluir la función de cálculo del descuento como una función de aparato, no de factura, así que deben modificarse los archivos originales para incluir esta nueva función.
Finalmente debe elaborar un programa cliente que permita la captura e impresión de tantas facturas como el usuario desee elaborar.

Diagrama de Clases



Código Fuente
¡Adelante! Ya saben como hacerlo…

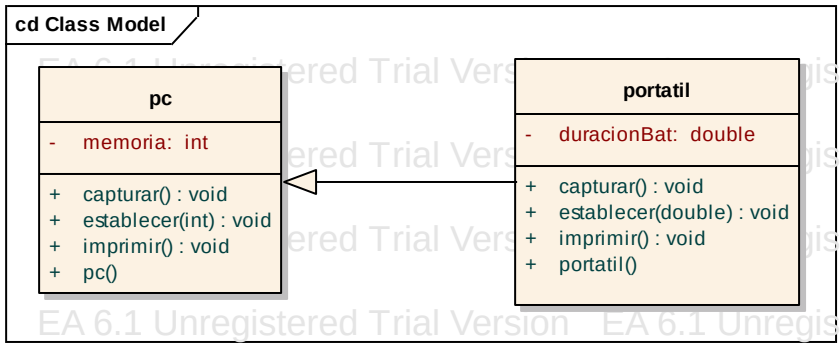
TERCER PARCIAL

Polimorfismo

Primer Ejemplo de Polimorfismo

Conceptos
Polimorfismo es la cualidad que tienen los objetos para responder de distinto modo a un mismo mensaje.
El polimorfismo nos permite escribir programas que procesen objetos de clases que formen parte de una misma jerarquía de clases, como si todos fueran objetos de la clase base.
De esta manera el número de métodos que tenemos que recordar al programar se reducen ostensiblemente.
A continuación un ejemplo en el que los métodos (funciones) de la clase portátil tienen exactamente los mismos nombres que las funciones equivalentes en la clase base (pc), y se demuestra en el código como pueden ser utilizadas ambas según se requiera, para objetos de tipo portátil

Diagrama de Clases



Código fuente
Pc.h
```
#ifndef pc_h
#define pc_h

class pc
{
private:
	int memoria;
public:
	pc();
	void establecer(int);
	void capturar();
	void imprimir();
};
class portatil : public pc
{
private:
	double duracionBat;
public:
	portatil();
	void establecer(double);
	void capturar();
	void imprimir();
};

#endif
```

Pc.cpp

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

pc::pc()
{
	establecer(128);
}

void pc::establecer(int m)
{
	memoria = m;
}

void pc::capturar()
{
	cout << "Ingrese tamaño de memoria: ";
	cin >> memoria;
}

void pc::imprimir()
{
	cout << "Tamaño de memoria: " << memoria << endl;
}

// Funciones de portatil
portatil::portatil()
{
}

void portatil::establecer(double d)
{
	duracionBat = d;
}

void portatil::capturar()
{
	/*pc::capturar();*/
	cout << "Duracion estimada de bateria (en minutos): ";
	cin >> duracionBat;
}
void portatil::imprimir()
{
	pc::imprimir();
	cout << "Duracion estimada de bateria (minutos): " << duracionBat << endl;
}
```

testPC.cpp

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

int main()
{
	pc *apuntadorPc;
	portatil compu;

	apuntadorPc = &compu;

	apuntadorPc->capturar();
	compu.capturar();
	compu.imprimir();

	system("pause");
	return 0;
}
```

Otro Ejemplo Polimorfismo: Clases Punto y Círculo

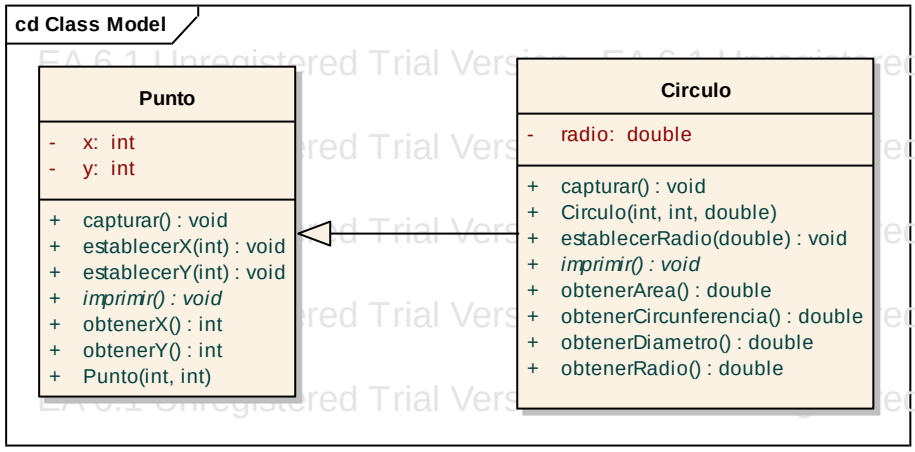
Planteamiento

Este es otro ejemplo que ilustra el tema de polimorfismo al utilizar dentro de una jerarquía de clases (**circulo** hereda de **punto**), funciones que tienen el mismo nombre.

Además, este ejercicio, en su programa cliente tiene como propósito determinar en cual cuadrante está el centro del círculo (el centro corresponde a los valores de **x** y **y**). Determinándose de la siguiente manera:

- Si x es menor que cero y y es mayor que cero, el centro está en el cuadrante 1.
- Si x es mayor que cero y y es mayor que cero, el centro está en el cuadrante 2.
- Si x es mayor que cero y y es menor que cero, el centro está en el cuadrante 3.
- Si x y y son ambas menores que cero, el centro está en el cuadrante 4.

Diagrama de Clases



Código fuente

Punto.h

```
// La definición de la clase Punto representa un par de coordenadas x-y.
#ifndef PUNTO_H
#define PUNTO_H

class Punto {

public:
   Punto( int = 0, int = 0 ); // constructor predeterminado

   void establecerX( int );  // establecer x en el par de coordenadas
   int obtenerX() const;  // devolver x del par de coordenadas

   void establecerY( int );  // establecer y en el par de coordenadas
   int obtenerY() const;  // devolver y del par de coordenadas

   // Agregada para este ejercicio
   void capturar();

   virtual void imprimir() const;  // mostrar objeto Punto

private:
   int x;  // la parte correspondiente a la x del par de coordenadas
   int y;  // la parte correspondiente a y del par de coordenadas

}; // fin de la clase Punto

#endif
```

Circulo.h

```
// La clase Circulo contiene un par de coordenadas x-y y el radio.
#ifndef CIRCULO_H
#define CIRCULO_H

#include "punto.h"  // Definición de la clase Punto

class Circulo : public Punto {
```

```
public:

   // constructor predeterminado
   Circulo( int = 0, int = 0, double = 0.0 );

   void establecerRadio( double );   // establecer el radio
   double obtenerRadio() const;   // devolver el radio

   double obtenerDiametro () const;       // devolver el diámetro
   double obtenerCircunferencia() const;  // devolver la circunferencia
   double obtenerArea() const;           // devolver el área

   void capturar();

   virtual void imprimir() const;         // mostrar el objeto Circulo

private:
   double radio;  // El radio del círculo

}; // fin de la clase Circulo

#endif
```

Punto.cpp

```
// Fig. 10.2: punto.cpp
// Definiciones de las funciones miembro de la clase Punto.
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

#include "punto.h"   // Definición de la clase Punto

// constructor predeterminado
Punto::Punto( int xValor, int yValor )
   : x( xValor ), y( yValor )
{
   // cuerpo vacío

} // fin de constructor de Punto

// establecer x en par de coordenadas
void Punto::establecerX( int xValor )
{
   x = xValor; // no hay necesidad de validación

} // fin de la función establecerX

// devolver x del par de coordenadas
int Punto::obtenerX() const
{
   return x;

} // fin de la función obtenerX

// establecer y en par de coordenadas
void Punto::establecerY( int yValor )
{
   y = yValor; // no hay necesidad de validación

} // fin de la función establecerY

// devolver y del par de coordenadas
int Punto::obtenerY() const
{
   return y;

} // fin de la función obtenerY

// mostrar objeto Punto
void Punto::imprimir() const
{
   cout << '[' << obtenerX() << ", " << obtenerY() << ']';

} // fin de la función imprimir

// Agregada para este ejercicio
// Capturar dato de un punto
void Punto::capturar()
{
        int vx = 0;
        int vy = 0;

        cout << "Valor de x: ";
        cin >> vx;
```

```
        cout << "Valor de y: ";
        cin >> vy;

        establecerY(vy);
        establecerX(vx);
}
```

Circulo.cpp

```
// Fig. 10.4: circulo.cpp
// Definiciones de las funciones miembro de la clase Circulo.
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

#include "circulo.h"   // Definición de la clase Circulo

// constructor predeterminado
Circulo::Circulo( int xValor, int yValor, double radioValor )
   : Punto( xValor, yValor )  // llamar al constructor de la clase base
{
   establecerRadio( radioValor );

} // fin de constructor de Circulo

// establecer el radio
void Circulo::establecerRadio( double radioValor )
{
   radio = ( radioValor < 0.0 ? 0.0 : radioValor );

} // fin de la función establecerRadio

// devolver el radio
double Circulo::obtenerRadio () const
{
   return radio;

} // fin de la función obtenerRadio

// calcular y devolver el diámetro
double Circulo::obtenerDiametro () const
{
   return 2 * obtenerRadio ();

} // fin de la función obtenerDiametro

// calcular y devolver la circunferencia
double Circulo::obtenerCircunferencia () const
{
   return 3.14159 * obtenerDiametro ();

} // fin de la función obtenerCircunferencia

// calcular y devolver el área
double Circulo::obtenerArea () const
{
   return 3.14159 * obtenerRadio() * obtenerRadio();

} // fin de la función obtenerArea

// mostrar objeto Circulo
void Circulo::imprimir() const
{
   cout << "centro = ";
   Punto::imprimir();  // invocar la función imprimir de Punto
   cout << "; radio = " << obtenerRadio ();

} // fin de la función imprimir

// Agregada para este ejercicio
// Funcion de captura de datos de círculo
void Circulo::capturar()
{
        Punto::capturar();

        double r = 0;

        cout << "Valor de radio: ";
        cin >> r;

        establecerRadio(r);
}
```

programaCliente.cpp

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

int main()
{
	int salir = 1;
	Circulo c1;
	Punto *ptr = 0;

	ptr = &c1;

	while(salir != 0)
	{
		c1.capturar();

		cout << "\nEl centro del círculo es: ";
		ptr->imprimir();

		cout << "\nLos datos completos del cicurlo son: ";
		c1.imprimir();
		cout << endl;

		// Decidiendo el cuadrante
		if ((c1.obtenerX() < 0) && (c1.obtenerY() > 0))
		{
			cout << "El centro está en el cuadrante 1" << endl;
		}
		if ((c1.obtenerX() > 0) && (c1.obtenerY() > 0))
		{
			cout << "El centro está en el cuadrante 2" << endl;
		}
		if ((c1.obtenerX() > 0) && (c1.obtenerY() < 0))
		{
			cout << "El centro está en el cuadrante 3" << endl;
		}
		if ((c1.obtenerX() < 0) && (c1.obtenerY() < 0))
		{
			cout << "El centro está en el cuadrante 4" << endl;
		}

		cout << "\nDesea Salir (SI=0)";
		cin >> salir;
	}

	system("pause");
}
```

Ejemplo combinado Herencia + Composición y Polimorfismo

Planteamiento

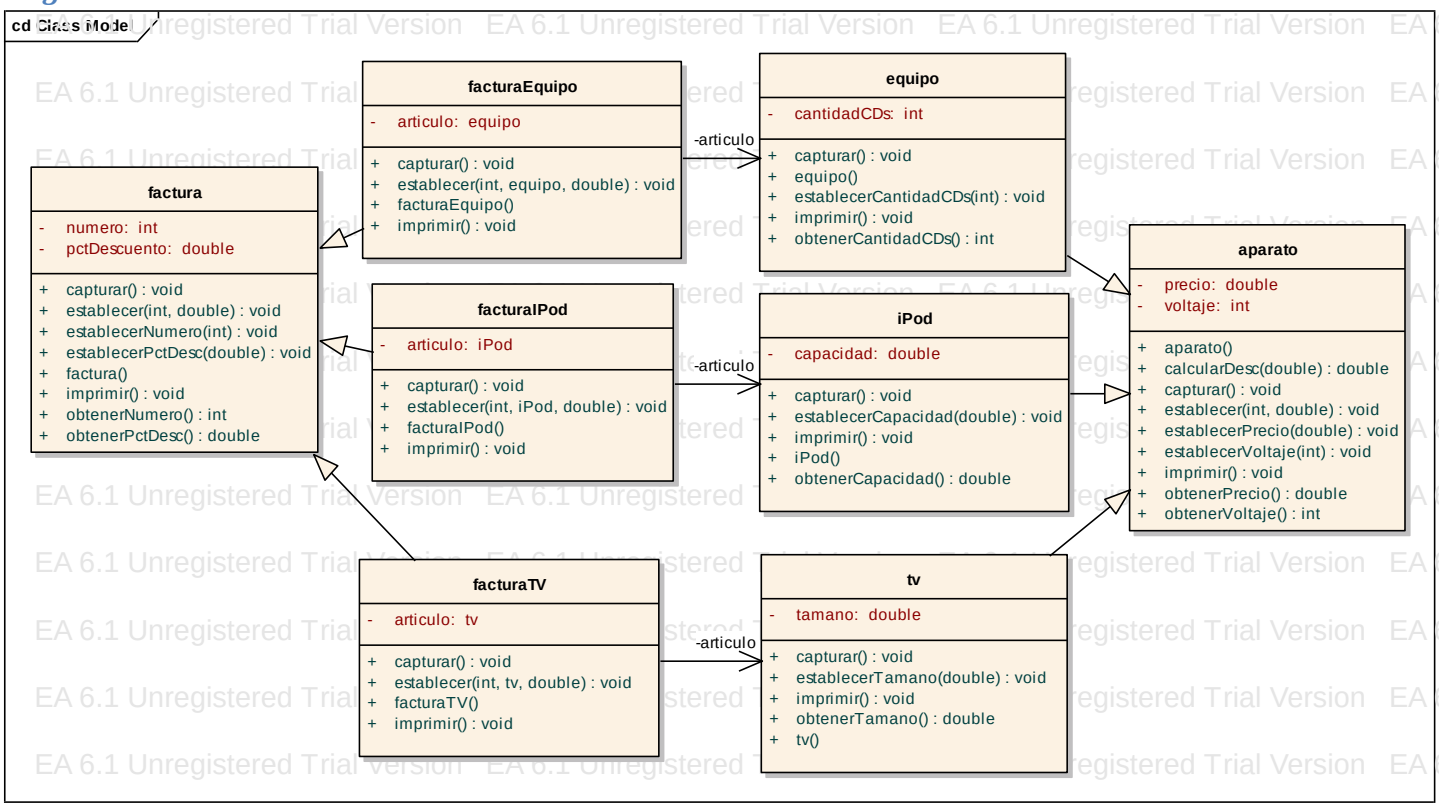
El siguiente ejemplo es uno de los más completos vistos ahora en este documento, en lo que respecta a la utilización de conceptos.

Tiene herencia: De aparato hacia una serie de aparatos electrónicos especializados y de factura a tipos especializados de factura.

Tiene composición: Las facturas especializadas tienen como datos miembro a objetos de los aparatos especializados.

Tiene polimorfismo: Todas las clases trabajan con funciones que llevan nombres genéricos.

Diagrama de clases



Código fuente

Electrónicos.h

```
#ifndef electronico_h
#define electronico_h
class aparato
{
private:
	int voltaje;
	double precio;
public:
	aparato();
	void establecer(int, double);
	void establecerVoltaje(int);
	void establecerPrecio(double);
	int obtenerVoltaje();
	double obtenerPrecio();
	void capturar();
	void imprimir();
	double calcularDesc(double);
};
class tv : public aparato
{
```

```
private:
	double tamano;
public:
	tv();
	void establecerTamano(double);
	double obtenerTamano();
	void capturar();
	void imprimir();
};
class equipo : public aparato
{
private:
	int cantidadCDs;
public:
	equipo();
	void establecerCantidadCDs(int);
	int obtenerCantidadCDs();
	void capturar();
	void imprimir();
};
class iPod : public aparato
{
private:
	double capacidad;
public:
	iPod();
	void establecerCapacidad(double);
	double obtenerCapacidad();
	void capturar();
	void imprimir();
};
#endif
```

Electrónicos.cpp

```
#include <iostream>
using namespace std;
#include "electronicos.h"
aparato::aparato()
{
	establecer(0, 0);
}
void aparato::establecer(int v, double p)
{
	establecerVoltaje(v);
	establecerPrecio(p);
}
void aparato::establecerVoltaje(int v)
{
	if (v > 0)
	{
		voltaje = v;
	}
	else
	{
		voltaje = 110;
	}
}
void aparato::establecerPrecio(double p)
{
	if (p > 0)
	{
		precio = p;
	}
	else
		precio = 0;
}
int aparato::obtenerVoltaje()
{
	return voltaje;
}
double aparato::obtenerPrecio()
{
	return precio;
}
void aparato::capturar()
{
	int v;
	double p;
	cout << "Voltaje: ";
	cin >> v;
	cout << "Precio: ";
	cin >> p;
	establecer(v, p);
}
void aparato::imprimir()
{
	cout << "Voltaje: " << voltaje << endl;
```

```
	cout << "Precio: " << obtenerPrecio() << endl;
}
double aparato::calcularDesc(double pct)
{
	return (precio * pct) / 100;
}

// Funciones de clase TV
tv::tv()
{
	establecerTamano(0);
}
void tv::establecerTamano(double t)
{
	if (t > 0)
	{
		tamano = t;
	}
	else
	{
		tamano = 0;
	}
}
double tv::obtenerTamano()
{
	return tamano;
}
void tv::capturar()
{
	cout << "Ingrese Datos del TV: " << endl;
	aparato::capturar();
	double t;
	cout << "Tamano: ";
	cin >> t;
	establecerTamano(t);
}
void tv::imprimir()
{
	cout << "Datos del TV: " << endl;
	aparato::imprimir();
	cout << "Tamano: " << tamano << endl;
}

// Funciones de la clase equipo
equipo::equipo()
{
	establecerCantidadCDs(1);
}
void equipo::establecerCantidadCDs(int c)
{
	if (c > 0)
	{
		cantidadCDs = c;
	}
	else
	{
		cantidadCDs = 1;
	}
}
int equipo::obtenerCantidadCDs()
{
	return cantidadCDs;
}
void equipo::capturar()
{
	int c;
	cout << "Ingrese datos del equipo: " << endl;
	aparato::capturar();
	cout << "Cantidad cd CDs: ";
	cin >> c;
	establecerCantidadCDs(c);
}
void equipo::imprimir()
{
	cout << "\nDatos del equipo: " << endl;
	aparato::imprimir();
	cout << "Cantidad CDs: " << obtenerCantidadCDs() << endl;
}

// Funciones de la clase iPod
iPod::iPod()
{
	establecerCapacidad(0);
}
void iPod::establecerCapacidad(double c)
{
	if (c > 0)
	{
```

```
                capacidad = c;
        }
        else
        {
                capacidad = 0;
        }
}
double iPod::obtenerCapacidad()
{
        return capacidad;
}
void iPod::capturar()
{
        double c;
        cout << "Ingrese datos del iPod: " << endl;
        aparato::capturar();
        cout << "Capacidad en Gigabytes: ";
        cin >> c;
        establecerCapacidad(c);
}
void iPod::imprimir()
{
        cout << "\nDatos del iPod: " << endl;
        aparato::imprimir();
        cout << "Capacidad: " << obtenerCapacidad() << endl;
}
```

Factura.h

```
#ifndef fact_h
#define fact_h
#include "electronicos.h"
class factura
{
private:
        int numero;
        double pctDescuento;
public:
        factura();
        void establecer(int, double);
        void establecerNumero(int);
        void establecerPctDesc(double);
        int obtenerNumero();
        double obtenerPctDesc();
        void capturar();
        void imprimir();
};
class facturaTV : public factura
{
private:
        tv articulo;
public:
        facturaTV();
        void establecer(int, tv, double);
        void capturar();
        void imprimir();
};
class facturaEquipo : public factura
{
private:
        equipo articulo;
public:
        facturaEquipo();
        void establecer(int, equipo, double);
        void capturar();
        void imprimir();
};
class facturaIPod : public factura
{
private:
        iPod articulo;
public:
        facturaIPod();
        void establecer(int, iPod, double);
        void capturar();
        void imprimir();
};
#endif
```

Factura.cpp

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

// Funciones de Clase Factura
factura::factura()
```

```
{
}
void factura::establecer(int n, double p)
{
	establecerNumero(n);
	establecerPctDesc(p);
}

void factura::establecerNumero(int n)
{
	if (n > 0)
	{
		numero = n;
	}
	else
	{
		numero = 0;
	}
}
void factura::establecerPctDesc(double p)
{
	if (p > 0)
	{
		pctDescuento = p;
	}
	else
	{
		pctDescuento = 0;
	}
}
int factura::obtenerNumero()
{
	return numero;
}
double factura::obtenerPctDesc()
{
	return pctDescuento;
}
void factura::capturar()
{
	int n = 0;
	double pct = 0;

	cout << "Capturando factura: " << endl;
	cout << "Numero: ";
	cin >> n;
	cout << "Porcentaje de descuento: " << endl;
	cin >> pct;

	establecer(n, pct);
}
void factura::imprimir()
{
	cout << "\n...imprimiendo datos de factura..." << endl;
	cout << "Numero: " << numero << endl;
	cout << "Porcentaje de descuento: " << pctDescuento << endl;
}

// Funciones de facturaTV
facturaTV::facturaTV()
{
}
void facturaTV::establecer(int n, tv t, double p)
{
	establecerNumero(n);
	establecerPctDesc(p);
	articulo = t;
}
void facturaTV::capturar()
{
	factura::capturar();
	articulo.capturar();
}
void facturaTV::imprimir()
{
	factura::imprimir();
	articulo.imprimir();
	cout << "Valor del descuento: "
		 << articulo.calcularDesc(obtenerPctDesc()) << endl;
	cout << "Neto a pagar: "
		<< articulo.obtenerPrecio() - articulo.calcularDesc(obtenerPctDesc());
	cout << endl;
}

//  Funciones para facturas de Equipos de Sonido
facturaEquipo::facturaEquipo()
{
}
```

```
void facturaEquipo::establecer(int n, equipo e, double p)
{
	establecerNumero(n);
	establecerPctDesc(p);
	articulo = e;
}
void facturaEquipo::capturar()
{
	factura::capturar();
	articulo.capturar();
}
void facturaEquipo::imprimir()
{
	factura::imprimir();
	articulo.imprimir();
	cout << "Valor del descuento: "
		 << articulo.calcularDesc(obtenerPctDesc()) << endl;
	cout << "Neto a pagar: "
		<< articulo.obtenerPrecio() - articulo.calcularDesc(obtenerPctDesc());
	cout << endl;
}

//  Funciones para facturas de Ipods
facturaIPod::facturaIPod()
{
}
void facturaIPod::establecer(int n, iPod i, double p)
{
	establecerNumero(n);
	establecerPctDesc(p);
	articulo = i;
}
void facturaIPod::capturar()
{
	factura::capturar();
	articulo.capturar();
}
void facturaIPod::imprimir()
{
	factura::imprimir();
	articulo.imprimir();
	cout << "Valor del descuento: "
		 << articulo.calcularDesc(obtenerPctDesc()) << endl;
	cout << "Neto a pagar: "
		<< articulo.obtenerPrecio() - articulo.calcularDesc(obtenerPctDesc());
	cout << endl;
}
```

programaFacturacion.cpp

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

int main()
{
	int salir = 0;
	while (salir != 1)
	{
		int ap;
		cout << "Cual aparato quiere facturar (1=TV, 2=Equipo, 3=iPod): ";
		cin >> ap;
		if (ap == 1)
		{
			facturaTV f;
			f.capturar();
			f.imprimir();
		}
		else
			if (ap == 2)
			{
				facturaEquipo f;
				f.capturar();
				f.imprimir();
			}
			else
			{
				facturaIPod f;
				f.capturar();
				f.imprimir();
			}
		cout << "Teclee 1 si desea salir, cualquier nro para seguir: ";
		cin >> salir;
	}
	system("pause");
}
```

Sobrecarga

Se utiliza sobrecarga de operadores cuando esto hace que el programa sea más claro que si llevara a cabo las mismas operaciones mediante llamadas explícitas a funciones.

Conceptualmente es correcto asumir que los operadores sobrecargados imitan la funcionalidad de sus contrapartes integradas; por ejemplo: el operador + debe sobrecargarse para realizar sumas, no restas.

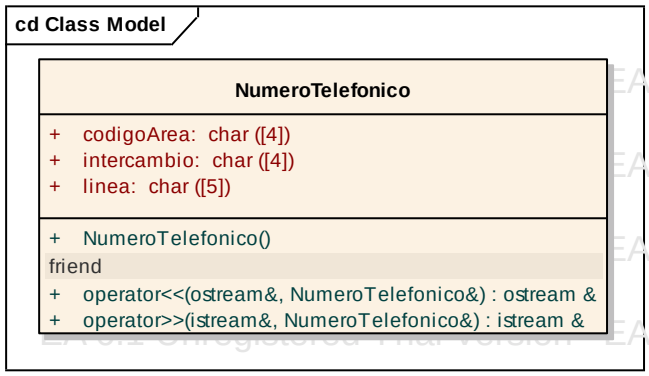
Los operadores que puede sobrecargarse en C++ son los siguientes: +, -, *, /. %, ^, &, |, ~, !, =, >, <, +=, -=, *=, /=, %=, ^=, &=, |=, <<, >>, >>=, <<=, ==, !=, <=, >=, &&, ||, ++, --, ->*, ->, [], (), new, delete, new[], delete[].

Primer Ejemplo Sobrecarga de operadores de inserción y extracción de flujo

Planteamiento

En este ejemplo se sobrecargan los operadores >> y <<, para perm tir la inserción y extracción de flujo de caracteres, de manera que puedan capturarse y mostrarse en pantalla los valores de los datos miembro de la clase teléfono como si fueran variables de tipos estándar de C++ (como *int* o *double*, por ejemplo).

Diagrama de Clase



Código fuente

telefonoSobrecargado.h

```
#ifndef sobrecarga_h
#define sobrecarga_h

class NumeroTelefonico
{
public:
        NumeroTelefonico();
        friend ostream &operator<<( ostream&, const NumeroTelefonico & );
        friend istream &operator>>( istream&,  NumeroTelefonico & );
        char codigoArea [4];
        char intercambio [4];
        char linea [5];
};
#endif
```

telefonoSobrecargado.cpp

```
#include <iostream>
using namespace std;
#include <iomanip>
using std::setw;
#include "telefonoSobreCargado.h"


NumeroTelefonico::NumeroTelefonico()
```

```
{
}

ostream &operator<<( ostream &salida, const NumeroTelefonico &num)
{
        salida << "(" << num.codigoArea << ")" << num.intercambio << "-"
               << num.linea << endl;
        return salida;
}
istream &operator>> (istream &entrada, NumeroTelefonico &num)
{
        entrada.ignore();
        entrada >> setw(4) >> num.codigoArea;
        entrada.ignore(2);
        entrada >> setw(4) >> num.intercambio;
        entrada.ignore();
        entrada >> setw(5) >> num.linea;

        return entrada;
}
```

testTelefono.cpp

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

int main()
{
        NumeroTelefonico tel;

        cout << "Ingrese numero. Por ejemplo: (504) 220-1122 : ";
        cin >> tel;

        cout << "Su numero es: " << tel << endl;

        system("PAUSE");
        return 0;
}
```

Segundo Ejemplo: Sobrecarga de Operadores de Incremento y Decremento

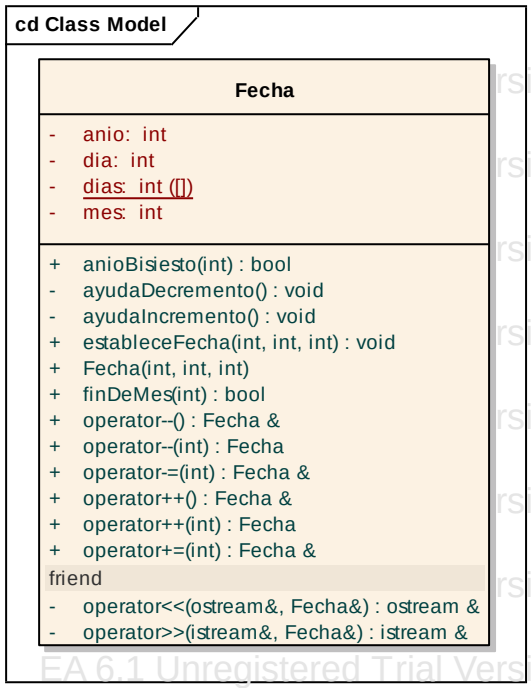
Planteamiento

Los operadores de incremento y decremento (++, --, += y -=) se utilizan para realizar sumas o restas a valores numéricos.

En este ejercicio, sin embargo, se implementará sobrecarga para que estos mismos operadores sirvan para sumarle o restarle días a objetos de tipo fecha.

Tenga en cuenta que, si bien los datos que componen una fecha son numéricos, la clase en sí no puede considerarse numérica, por lo que los operadores no pueden ser utilizados directamente, a menos que se programen las funciones sobrecargadas.

Diagrama de clases



Código fuente

Fecha.h

```
#ifndef FECHA1_H
#define FECHA1_H
#include <iostream>
using std::ostream;
using std::istream;

class Fecha {
   // Sobrecargando operadores de inserción y extracción de flujo
        friend ostream &operator<<( ostream&, const Fecha & );
        friend istream &operator>>( istream&, Fecha & );
public:
        // Constructor y función establece
   Fecha( int m = 1, int d = 1, int y = 1900 ); // constructor
   void estableceFecha( int, int, int ); // establece la fecha
   // Sobrecargando operadores de incremento y decremento
   Fecha &operator++();             // operador de preincremento
   Fecha operator++( int );         // operador de postincremento
   Fecha &operator--();             // operador de predecremento
   Fecha operator--( int );         // operador de postdecremento
   const Fecha &operator+=( int ); // suma días, modifica el objeto
   const Fecha &operator-=( int ); // resta días, modifica el objeto
```

```
   // Funciones de utilidad que actúan sobre objetos de tipo Fecha
   bool anioBisiesto( int ) const;  // ¿es año bisiesto?
   bool finDeMes( int ) const;  // ¿es éste el fin de mes?

private:
   // Datos miembro de la clase
        int mes;
        int dia;
        int anio;
   // Arreglo de valores constantes para saber cuantos días tiene el mes de la fecha
        static const int dias[];       // arreglo de días por mes
   // Funciones que suman o restan un día a objetos de tipo fecha
   void ayudaIncremento();        // función de utilidad
   void ayudaDecremento();
}; // fin de la clase Fecha
#endif
```

Fecha.cpp

```
// Definición de las funciones miembro para la clase Fecha.
#include <iostream>
#include <iomanip>
using std::setw;
#include "fecha.h"

// inicializa el miembro estático con alcance de archivo;
// una copia propia de la clase
const int Fecha::dias[] =
   { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

// constructor Fecha
Fecha::Fecha( int m, int d, int a )
{
   estableceFecha( m, d, a );
} // fin del constructor Fecha

// establece mes, día y año
void Fecha::estableceFecha( int mm, int dd, int aa )
{
   mes = ( mm >= 1 && mm <= 12 ) ? mm : 1;
   anio = ( aa >= 1900 && aa <= 2100 ) ? aa : 1900;

   // verifica si es año bisiesto
   if ( mes == 2 && anioBisiesto( anio ) )
      dia = ( dd >= 1 && dd <= 29 ) ? dd : 1;
   else
      dia = ( dd >= 1 && dd <= dias[ mes ] ) ? dd : 1;
} // fin de la función estableceFecha

// FUNCIONES DE INCREMENTO
// operador de preincremento sobrecargado
Fecha &Fecha::operator++()
{
   ayudaIncremento();
   return *this;  // devuelve una referencia para crear un lvalue
} // fin de la función operator++

// Función para el operador de posincremento sobrecargado; observe que el parámetro
// entero falso no tiene nombre de parámetro, solo sirve para diferenciar
Fecha Fecha::operator++( int )
{
   Fecha temp = *this;
   ayudaIncremento();  // mantiene el estado actual del objeto
   // devuelve el objeto temporal sin incremento guardado
   return temp;   // valor de retorno; no es una referencia de retorno
} // fin de la función operator++

// esta función suma el número especificado de días a una fecha
const Fecha &Fecha::operator+=( int diasAdicionales )
{
   for ( int i = 0; i < diasAdicionales; i++ )
      ayudaIncremento();
   return *this;    // permite la cascada
} // fin de la función operator+=

// FUNCIONES DE DECREMENTO
// operador de predecremento sobrecargado
Fecha &Fecha::operator--()
{
   ayudaDecremento();
   return *this;  // devuelve una referencia para crear un lvalue
} // fin de la función operator++

// Función para el operador de posincremento sobrecargado; observe que el parámetro
// entero falso no tiene nombre de parámetro, solo sirve para diferenciar
Fecha Fecha::operator--( int )
```

```
{
   Fecha temp = *this;
   ayudaDecremento();  // mantiene el estado actual del objeto
   // devuelve el objeto temporal sin incremento guardado
   return temp;   // valor de retorno; no es una referencia de retorno
} // fin de la función operator++

// esta función suma el número especificado de días a una fecha
const Fecha &Fecha::operator-=( int diasARestar )
{
   for ( int i = 0; i < diasARestar; i++ )
      ayudaDecremento();
   return *this;    // permite la cascada
} // fin de la función operator+=


// si el año es bisiesto, devuelve verdadero;
// de lo contrario, devuelve falso
bool Fecha::anioBisiesto( int verificaAnio ) const
{
   if ( verificaAnio % 400 == 0 ||
      ( verificaAnio % 100 != 0 && verificaAnio % 4 == 0 ) )
      return true;   // anio bisiesto
   else
      return false;  // no es anio bisiesto
} // fin de la función anioBisiesto

// determina si el día es el último del mes
bool Fecha::finDeMes( int verificaDia ) const
{
   if ( mes == 2 && anioBisiesto( anio ) )
      return verificaDia == 29; // último día de Feb. en año bisiesto
   else
      return verificaDia == dias[ mes ];
} // fin de la función finDeMes

// FUNCION DE INCREMENTO
// función de utilidad para ayudar a incrementar la fecha en un día
void Fecha::ayudaIncremento()
{
   // el día no es fin de mes
   if ( !finDeMes( dia ) )
      ++dia;
   else
      // el día es fin de mesy mes es < 12
      if ( mes < 12 ) {
         ++mes;
         dia = 1;
      }
      // último día del mes
      else {
         ++anio;
         mes = 1;
         dia = 1;
      }
} // fin de la función ayudaIncremento

// FUNCION DE DECREMENTO
// función de utilidad para ayudar a decrementar la fecha en un día
void Fecha::ayudaDecremento()
{
   // Se resta un día, si el día no es el primero del mes
   if ( dia > 1 )
      --dia;
   else
      // Se resta un mes si el día es inicio de mes
         // y mes no es enero
      if ( mes > 1 ) {
         --mes;
         dia = dias[ mes ];
      }
      // Se resta un año si es uno de enero
      else {
         --anio;
         mes = 12;
         dia = 31;
      }
} // fin de la función ayudaDecremento

// operador de salida (inserción de flujo) sobrecargado
// Muestra la fecha en Letras
ostream &operator<<( ostream &salida, const Fecha &d )
{
   static char *nombreMes[ 13 ] = { "", "Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio", "Julio",
"Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre" };

   salida << nombreMes[ d.mes ] << ' ' << d.dia << ", " << d.anio;
```

```
   return salida;   // permite la cascada
} // fin de la función operator<<


// operador de entrada (extracción de flujo) sobrecargado

istream &operator>>( istream &entrada, Fecha &f )
{
   entrada >> setw( 2 ) >> f.dia;  // introduce el dia
   entrada.ignore( 1 );           // evita la /
   entrada >> setw( 2 ) >> f.mes;  // introduce el mes
   entrada.ignore(1);             // evita la /
   entrada >> setw( 4 ) >> f.anio; // introduce el año

   return entrada;      // habilita cin >> a >> b >> c;

} // fin de la función operator>>
```

probandoOperadoresFecha.cpp

```
// Programa de prueba para la clase Fecha.
#include <iostream>
using std::cout;
using std::cin;
using std::endl;
#include "fecha.h"  // definición de la clase Fecha

int main()
{
   Fecha f1;
   Fecha f2(12,27,1992);
       Fecha f3( 0, 99, 8045 );  // fecha no válida

   cout << "f1 es " << f1 << "\nf2 es " << f2 << "\nf3 es " << f3;
   cout << "\n\nf2 += 7 es " << ( f2 += 7 );

   f3.estableceFecha( 2, 28, 1993 );
   cout << "\n\n  f3 es " << f3;
   cout << "\n++f3 es " << ++f3;

   Fecha f4;
   cout << "\nIngrese una fecha con formato dd/mm/aaaa: ";
   cin >> f4;

   cout << "\n\nPrueba del operador de preincremento:\n" << "  f4 es " << f4 << '\n';
   cout << "++f4 es " << ++f4 << '\n';
   cout << "  f4 es " << f4;
   cout << "\n\nPrueba del operador de posincremento:\n" << "  f4 es " << f4 << '\n';
   cout << "f4++ es " << f4++ << '\n';
   cout << "  f4 es " << f4 << endl;

   cout << "\n------- Probando los operadores de decremento -------" << endl;
   cout << "Valor actual de f4: " << f4 << endl;
   cout << "Restandole un día a la fecha (predecremento): " << --f4 << endl;
   cout << "Restandole un dia a la fecha (posdecremento): " << f4-- << endl;
   cout << "Ahora si miremos el efecto: " << f4 << endl;
   cout << "Restandole 5 dias a f4: " << (f4-= 5) << endl;
   cout << "Restandole otros 10 dias: " << (f4-= 10) << endl;
   f4.estableceFecha(1, 2, 2005);
   cout << "Estableciendo nueva fecha para f4: " << f4 << endl;
   cout << "Restandole 4 dias: " << (f4-= 4) << endl;

   system("pause");
   return 0;
} // fin de main
```

Ejemplo Final: Proyecto Préstamos

Primer ejercicio

Planteamiento del problema

Se requiere un programa para la registro de préstamos en una cooperativa.
Lo datos que se manejan para el préstamo son los siguientes:

- Número de Préstamo (numérico entero)
- Solicitante del préstamo (Persona). Se requiere únicamente: Nro. De identidad, Primer Nombre, Primer y Segundo Apellido, teléfono de casa y teléfono móvil.
- Valor del préstamo (numérico con decimales)
- Fechas de pago de las cuotas (arreglo de un máximo de 6 fechas, se asume que el plazo máximo de pago son 6 meses).
- Fecha de autorización del préstamo.
- Fecha tentativa de entrega del préstamo.

Las reglas que debe respetar este proyecto son las siguientes:

- El número de préstamo siempre deberá ser un valor mayor que cero.
- El valor del préstamo siempre debe ser mayor a cero.
- Debe haber una función de captura de los datos del solicitante debe capturar únicamente los datos requeridos.
- La fecha tentativa de entrega del préstamo será siete días después de la fecha de autorización del préstamo.
- Las fechas de pago del préstamo se calculan, sumando 30 días a cada una a partir de la fecha de entrega del préstamo.
- Los préstamos solo se pueden autorizar en los primeros 20 días del mes. Esta es una política que nunca va a cambiar.

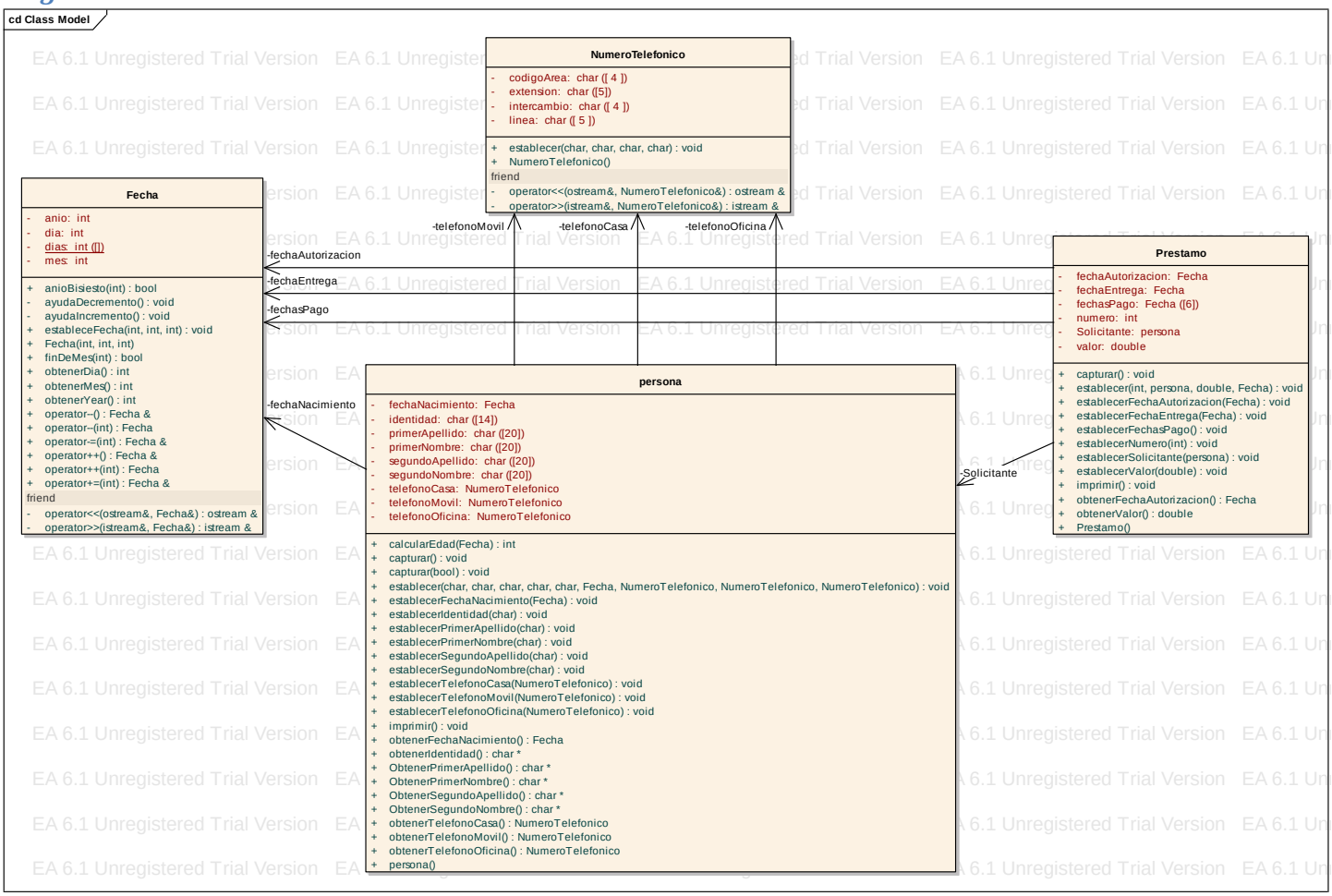
Las reglas técnicas a respetar en este ejercicio son las siguientes:

- Debe utilizar funciones polimórficas para las funciones establecer, en caso de que se implemente herencia.
- Debe utilizar operadores sobrecargados para la captura de datos de tipo fecha.
- Debe utilizar operadores sobrecargados para el cálculo de las fechas de pago y de entrega del préstamo.

El programa cliente de la clase debe reunir los siguientes requerimientos:

- Existe una fecha máxima para la autorización de los préstamos.
- Existe un gran valor máximo a prestar. La sumatoria de los préstamos que se ingresen no debe exceder este valor.
- Debe permitir la captura de tantos préstamos como desee ingresar el usuario, a menos que se haya llegado al valor máximo a prestar.
- Antes de capturar el préstamo debe preguntar si se desea capturar los datos completos del solicitante o únicamente los datos requeridos por el préstamo.
- Debe imprimir los datos completos del préstamo, incluyendo la fecha de entrega y las fechas de pago de las cuotas.

Diagrama de Clases



Código fuente

Fecha.h

```
#ifndef FECHA1_H
#define FECHA1_H
#include <iostream>
using std::ostream;
using std::istream;
class Fecha {
   // Sobrecargando operadores de inserción y extracción de flujo
        friend ostream &operator<<( ostream&, const Fecha & );
        friend istream &operator>>( istream&, Fecha & );
public:
        // Constructor y función establece
   Fecha( int m = 1, int d = 1, int y = 1900 ); // constructor
   void estableceFecha( int, int, int ); // establece la fecha
   // Funciones obtener para clase Fecha
   int obtenerDia();
   int obtenerMes();
   int obtenerYear();
   // Sobrecargando operadores de incremento y decremento
   Fecha &operator++();            // operador de preincremento
   Fecha operator++( int );        // operador de postincremento
   Fecha &operator--();            // operador de predecremento
   Fecha operator--( int );        // operador de postdecremento
   const Fecha &operator+=( int ); // suma días, modifica el objeto
   const Fecha &operator-=( int ); // resta días, modifica el objeto
   // Funciones de utilidad que actúan sobre objetos de tipo Fecha
   bool anioBisiesto( int ) const;  // ¿es año bisiesto?
   bool finDeMes( int ) const;  // ¿es éste el fin de mes?
private:
   // Datos miembro de la clase
```

```
        int mes;
        int dia;
        int anio;
   // Arreglo de valores constantes para saber cuantos días tiene el mes de la fecha
        static const int dias[];       // arreglo de días por mes
   // Funciones que suman o restan un día a objetos de tipo fecha
   void ayudaIncremento();        // función de utilidad
   void ayudaDecremento();
}; // fin de la clase Fecha
#endif
```

Teléfono.h

```
#ifndef telefono_h
#define telefono_h
#include <iostream>
using std::ostream;
using std::istream;
// definición de la clase NumeroTelefonico
class NumeroTelefonico
{
   friend ostream &operator<<( ostream&, const NumeroTelefonico & );
   friend istream &operator>>( istream&, NumeroTelefonico & );
public:
        // Constructor
        NumeroTelefonico();
        // Función para establecer nuevo número Telefónico
        void establecer (char [4], char [4], char [5], char [5]);
private:
   char codigoArea[ 4 ];  // 3 dígitos para el código de área y el nulo
   char intercambio[ 4 ]; // 3 dígitos para intercambio y el nulo
   char linea[ 5 ];       // 4 dígitos para el código de área y el nulo
   char extension [5]; // 4 dígitos para el número y el nulo
}; // fin de la clase NumeroTelefonico
#endif
```

Persona.h

```
#ifndef clases_persona_h
#define clases_persona_h
#include "fecha.h"
#include "telefono.h"
class persona
{
private:
        char identidad [14];
        char primerNombre [20];
        char segundoNombre [20];
        char primerApellido [20];
        char segundoApellido [20];
        Fecha fechaNacimiento;
        NumeroTelefonico telefonoCasa;
        NumeroTelefonico telefonoOficina;
        NumeroTelefonico telefonoMovil;
public:
        persona();
        void establecer (char [14], char [20], char [20], char [20], char [20], Fecha, NumeroTelefonico,
NumeroTelefonico, NumeroTelefonico);
        void establecerIdentidad (char [20]);
        void establecerPrimerNombre (char [20]);
        void establecerSegundoNombre (char [20]);
        void establecerPrimerApellido (char [20]);
        void establecerSegundoApellido (char [20]);
        void establecerFechaNacimiento (Fecha);
        void establecerTelefonoCasa(NumeroTelefonico);
        void establecerTelefonoOficina(NumeroTelefonico);
        void establecerTelefonoMovil(NumeroTelefonico);
        char *obtenerIdentidad();
        char *ObtenerPrimerNombre();
        char *ObtenerSegundoNombre();
        char *ObtenerPrimerApellido();
        char *ObtenerSegundoApellido();
        Fecha obtenerFechaNacimiento();
        NumeroTelefonico obtenerTelefonoCasa();
        NumeroTelefonico obtenerTelefonoOficina();
        NumeroTelefonico obtenerTelefonoMovil();
        void capturar();
        void capturar(bool);
        void imprimir();
        int calcularEdad( Fecha);
};
#endif
```

Prestamo.h

```
#ifndef prestamo_h
#define prestamo_h
```

```
#include "fecha.h"
#include "persona.h"

class Prestamo
{
public:
        Prestamo();
        void establecer(int, persona, double, Fecha);
        void establecerNumero(int);
        void establecerSolicitante(persona);
        void establecerValor(double);
        void establecerFechaAutorizacion(Fecha);
        void establecerFechaEntrega(Fecha);
        void establecerFechasPago();
        void capturar();
        void imprimir();
        // agregado
        Fecha obtenerFechaAutorizacion();
        double obtenerValor();
private:
        int numero;
        persona Solicitante;
        double valor;
        Fecha fechaAutorizacion;
        Fecha fechaEntrega;
        Fecha fechasPago [6];
};
#endif
```

Fecha.cpp

```
// Fecha.cpp
// Definición de las funciones miembro para la clase Fecha.
#include <iostream>
#include <iomanip>
using std::setw;
#include "fecha.h"

// inicializa el miembro estático con alcance de archivo;
// una copia propia de la clase
const int Fecha::dias[] =  { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
// constructor Fecha
Fecha::Fecha( int m, int d, int a )
{
   estableceFecha( m, d, a );
} // fin del constructor Fecha

// establece mes, día y año
void Fecha::estableceFecha( int mm, int dd, int aa )
{
   mes = ( mm >= 1 && mm <= 12 ) ? mm : 1;
   anio = ( aa >= 1900 && aa <= 2100 ) ? aa : 1900;
   // verifica si es año bisiesto
   if ( mes == 2 && anioBisiesto( anio ) )
      dia = ( dd >= 1 && dd <= 29 ) ? dd : 1;
   else
      dia = ( dd >= 1 && dd <= dias[ mes ] ) ? dd : 1;
} // fin de la función estableceFecha

// FUNCIONES OBTENER
int Fecha::obtenerDia()
{       return dia; }
int Fecha::obtenerMes()
{       return mes; }

int Fecha::obtenerYear()
{       return anio; }

// FUNCIONES DE INCREMENTO
// operador de preincremento sobrecargado
Fecha &Fecha::operator++()
{
   ayudaIncremento();
   return *this;  // devuelve una referencia para crear un lvalue
} // fin de la función operator++
// Función para el operador de posincremento sobrecargado; observe que el parámetro
// entero falso no tiene nombre de parámetro, solo sirve para diferenciar
Fecha Fecha::operator++( int )
{
   Fecha temp = *this;
   ayudaIncremento();  // mantiene el estado actual del objeto
   // devuelve el objeto temporal sin incremento guardado
   return temp;   // valor de retorno; no es una referencia de retorno
} // fin de la función operator++
// esta función suma el número especificado de días a una fecha
const Fecha &Fecha::operator+=( int diasAdicionales )
{
```

```
   for ( int i = 0; i < diasAdicionales; i++ )
      ayudaIncremento();
   return *this;     // permite la cascada
} // fin de la función operator+=

// FUNCIONES DE DECREMENTO
// operador de predecremento sobrecargado
Fecha &Fecha::operator--()
{
   ayudaDecremento();
   return *this;  // devuelve una referencia para crear un lvalue
} // fin de la función operator++
// Función para el operador de posincremento sobrecargado; observe que el parámetro
// entero falso no tiene nombre de parámetro, solo sirve para diferenciar
Fecha Fecha::operator--( int )
{
   Fecha temp = *this;
   ayudaDecremento();  // mantiene el estado actual del objeto
   // devuelve el objeto temporal sin incremento guardado
   return temp;   // valor de retorno; no es una referencia de retorno
} // fin de la función operator++
// esta función suma el número especificado de días a una fecha
const Fecha &Fecha::operator-=( int diasARestar )
{
   for ( int i = 0; i < diasARestar; i++ )
      ayudaDecremento();
   return *this;     // permite la cascada
} // fin de la función operator+=

// si el año es bisiesto, devuelve verdadero;
// de lo contrario, devuelve falso
bool Fecha::anioBisiesto( int verificaAnio ) const
{
   if ( verificaAnio % 400 == 0 ||
      ( verificaAnio % 100 != 0 && verificaAnio % 4 == 0 ) )
      return true;   // anio bisiesto
   else
      return false;  // no es anio bisiesto
} // fin de la función anioBisiesto

// determina si el día es el último del mes
bool Fecha::finDeMes( int verificaDia ) const
{
   if ( mes == 2 && anioBisiesto( anio ) )
      return verificaDia == 29; // último día de Feb. en año bisiesto
   else
      return verificaDia == dias[ mes ];
} // fin de la función finDeMes

// FUNCION DE INCREMENTO
// función de utilidad para ayudar a incrementar la fecha en un día
void Fecha::ayudaIncremento()
{
   // el día no es fin de mes
   if ( !finDeMes( dia ) )
      ++dia;
   else
      // el día es fin de mesy mes es < 12
      if ( mes < 12 ) {
         ++mes;
         dia = 1;
      }
      // último día del mes
      else {
         ++anio;
         mes = 1;
         dia = 1;
      }
} // fin de la función ayudaIncremento

// FUNCION DE DECREMENTO
// función de utilidad para ayudar a decrementar la fecha en un día
void Fecha::ayudaDecremento()
{
   // Se resta un día, si el día no es el primero del mes
   if ( dia > 1 )
      --dia;
   else
      // Se resta un mes si el día es inicio de mes
       // y mes no es enero
      if ( mes > 1 ) {
         --mes;
         dia = dias[ mes ];
      }
      // Se resta un año si es uno de enero
      else {
         --anio;
         mes = 12;
```

```
         dia = 31;
      }
} // fin de la función ayudaDecremento
// operador de salida (inserción de flujo) sobrecargado
// Muestra la fecha en Letras
ostream &operator<<( ostream &salida, const Fecha &d )
{
   static char *nombreMes[ 13 ] = { "", "Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio", "Julio",
"Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre" };
   salida << nombreMes[ d.mes ] << ' ' << d.dia << ", " << d.anio;
   return salida;   // permite la cascada
} // fin de la función operator<<

// operador de entrada (extracción de flujo) sobrecargado
istream &operator>>( istream &entrada, Fecha &f )
{
   entrada >> setw( 2 ) >> f.dia;  // introduce el dia
   entrada.ignore( 1 );           // evita la /
   entrada >> setw( 2 ) >> f.mes;  // introduce el mes
   entrada.ignore(1);             // evita la /
   entrada >> setw( 4 ) >> f.anio; // introduce el año
   return entrada;      // habilita cin >> a >> b >> c;
} // fin de la función operator>>
```

Teléfono.cpp

```
// telefono.cpp
// Funciones miembro para clase NumeroTelefonico
#include <iostream>
using std::cout;
using std::cin;
using std::endl;
using std::ostream;
using std::istream;
#include <iomanip>
using std::setw;
#include "telefono.h"
// Constructor
NumeroTelefonico::NumeroTelefonico()
{       establecer(" ", " ", " ", " ");
}
// Función Establecer
void NumeroTelefonico::establecer(char a[4], char i[4], char l[5], char e[5])
{
        strcpy_s(codigoArea, a);
        strcpy_s(intercambio, i);
        strcpy_s(linea, l);
        strcpy_s(extension, e);
}
// operador de inserción de flujo sobrecargado; no puede ser
// una función miembro si deseamos invocarla con
// cout << algunNumeroTelefonico;
ostream &operator<<( ostream &salida, const NumeroTelefonico &num )
{
   salida << "(" << num.codigoArea << ") "  << num.intercambio << "-" << num.linea
              << " Ext. " << num.extension;
   return salida;     // habilita cout << a << b << c;
} // fin de la función operator<<

// operador extracción de flujo sobrecargado; no puede ser
// una función miembro si deseamos invocarla con
// cin >> algunNumeroTelefonico;
istream &operator>>( istream &entrada, NumeroTelefonico &num )
{
        entrada.ignore(2);                          // evita el (
   entrada >> setw( 4 ) >> num.codigoArea;  // introduce el código de área
   entrada.ignore( 2 );                     // evita el ) y el espacio
   entrada >> setw( 4 ) >> num.intercambio; // introduce el intercambio
   entrada.ignore();                        // evita el guión (-)
   entrada >> setw( 5 ) >> num.linea;        // introduce línea
   entrada.ignore(5);                        // evita la palabra ext. y espacio
   entrada >> setw( 5 ) >> num.extension;        // introduce nro. de extensión
   return entrada;      // habilita cin >> a >> b >> c;
} // fin de la función operator>>
```

Persona.cpp

```
#include <iostream>
using namespace std;
#include "fecha.h"
#include "telefono.h"
#include "persona.h"
// Clase Persona
persona::persona()
{
        char i [14];
        char n1 [20];
```

```
        char n2 [20];
        char a1 [20];
        char a2 [20];
        strcpy_s(i, "");
        strcpy_s(n1, "");
        strcpy_s(n2, "");
        strcpy_s(a1, "");
        strcpy_s(a2, "");
        Fecha f1;
        NumeroTelefonico tcasa;
        NumeroTelefonico tofi;
        NumeroTelefonico tmov;
        establecer(i, n1, n2, a1, a2, f1, tcasa, tofi, tmov);
}

void persona::establecer(char i [14], char n1 [20], char n2 [20], char a1 [20], char a2 [20], Fecha f,
NumeroTelefonico tcasa, NumeroTelefonico tofi, NumeroTelefonico tmov)
{
        establecerIdentidad(i);
        establecerPrimerNombre(n1);
        establecerSegundoNombre(n2);
        establecerPrimerApellido(a1);
        establecerSegundoApellido(a2);
        establecerFechaNacimiento(f);
        establecerTelefonoCasa(tcasa);
        establecerTelefonoOficina(tofi);
        establecerTelefonoMovil(tmov);
}

void persona::establecerIdentidad(char id [14])
{       strcpy_s(identidad, id); }
void persona::establecerPrimerNombre(char pn [20])
{       strcpy_s(primerNombre, pn); }
void persona::establecerSegundoNombre(char sn [20])
{       strcpy_s(segundoNombre, sn); }
void persona::establecerPrimerApellido(char pa [20])
{       strcpy_s(primerApellido, pa); }
void persona::establecerSegundoApellido(char sa [20])
{       strcpy_s(segundoApellido, sa); }
void persona::establecerFechaNacimiento(Fecha f)
{       fechaNacimiento = f; }
void persona::establecerTelefonoCasa(NumeroTelefonico tc)
{       telefonoCasa = tc; }
void persona::establecerTelefonoOficina(NumeroTelefonico to)
{       telefonoOficina = to; }
void persona::establecerTelefonoMovil(NumeroTelefonico tm)
{       telefonoMovil = tm; }
char *persona::obtenerIdentidad()
{       return identidad; }

char *persona::ObtenerPrimerNombre()
{       return primerNombre; }

char *persona::ObtenerSegundoNombre()
{       return segundoNombre; }
char *persona::ObtenerPrimerApellido()
{       return primerApellido; }
char *persona::ObtenerSegundoApellido()
{       return segundoApellido; }
Fecha persona::obtenerFechaNacimiento()
{       return fechaNacimiento; }
NumeroTelefonico persona::obtenerTelefonoCasa()
{       return telefonoCasa; }

NumeroTelefonico persona::obtenerTelefonoOficina()
{       return telefonoOficina; }
NumeroTelefonico persona::obtenerTelefonoMovil()
{       return telefonoMovil; }

void persona::capturar()
{
        char id [14];
        char pn [20];
        char sn [20];
        char pa [20];
        char sa [20];
        Fecha f;
        NumeroTelefonico tcasa;
        NumeroTelefonico tofi;
        NumeroTelefonico tmov;
        cout << "\n---------------- Capturando datos de una persona ----------------" << endl;
        cout << "Identidad: ";
        cin.ignore();
        cin.getline(id, 14);
        cout << "Primer Nombre: ";
        cin.getline(pn, 20);
        cout << "Segundo Nombre: ";
        cin.getline(sn, 20);
```

```
        cout << "Primer Apellido: ";
        cin.getline(pa, 20);
        cout << "Segundo Apellido: ";
        cin.getline(sa, 20);
        cout << "Fecha de Nacimiento (dd/mm/aaaa):\n";
        cin >> f;
        cout << "-- Para numeros de telefono utilice este formato: (xxx) xxx-xxxx ext.xxx --\n" ;
        cout << "Telefono de Casa:\n";
        cin >> tcasa;
        cout << "Telefono de Oficina:\n";
        cin >> tofi;
        cout << "Telefono Movil:\n";
        cin >> tmov;
        establecer(id, pn, sn, pa, sa, f, tcasa, tofi, tmov);
}

void persona::imprimir()
{
        cout << "\n------------- Imprimiendo datos personales --------------" << endl;
        cout << "Identidad: " << identidad << endl;
        cout << "Primer Nombre: " << primerNombre << endl;
        cout << "Segundo Nombre: " << segundoNombre << endl;
        cout << "Primer Apellido: " << primerApellido << endl;
        cout << "Segundo Apellido: " << segundoApellido << endl;
        cout << "Fecha de Nacimiento: " <<   fechaNacimiento << endl;
        cout << "Telefono de Casa: " << telefonoCasa << endl;
        cout << "Telefono de Oficina: " <<   telefonoOficina << endl;
        cout << "Telefono Movil: " << telefonoMovil << endl;
}


int persona::calcularEdad(Fecha fActual)
{
        int edad;
        if (fActual.obtenerMes() > fechaNacimiento.obtenerMes())
        {
                edad = fActual.obtenerYear() - fechaNacimiento.obtenerYear();
        }
        else
                if (fActual.obtenerMes() < fechaNacimiento.obtenerMes())
                {
                        edad = fActual.obtenerYear() - fechaNacimiento.obtenerYear() - 1;
                }
                else
                        if (fActual.obtenerDia() < fechaNacimiento.obtenerDia())
                        {
                                edad = fActual.obtenerYear() - fechaNacimiento.obtenerYear() - 1;
                        }
                        else
                        {
                                edad = fActual.obtenerYear() - fechaNacimiento.obtenerYear();
                        };
        return edad;
}
```

Préstamo.cpp

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

Prestamo::Prestamo()
{       int nro = 0;
        persona sol;
        double val = 0;
        Fecha fa;
        establecer(nro, sol, val, fa);
}
void Prestamo::establecer(int n, persona s, double v, Fecha fa)
{       establecerNumero(n);
        establecerSolicitante(s);
        establecerValor(v);
        establecerFechaAutorizacion(fa);
        establecerFechaEntrega(fa);
        establecerFechasPago();
}
void Prestamo::establecerNumero(int n)
{
        if (n > 0)
        {       numero = n;
        }
        else
        {       numero = 1;
        }
}
void Prestamo::establecerSolicitante(persona s)
{
        Solicitante = s;
```

```
}
void Prestamo::establecerValor(double v)
{
	if (v > 0)
	{	valor = v;
	}
	else
	{	valor = 0;
	}
}
void Prestamo::establecerFechaAutorizacion(Fecha fa)
{	fechaAutorizacion = fa;
}
void Prestamo::establecerFechaEntrega(Fecha fa)
{	fechaEntrega = (fa+= 7);
}
void Prestamo::establecerFechasPago()
{
	Fecha fechaTrabajo = fechaEntrega;
	for (int i = 0; i < 6; i++)
	{
		fechaTrabajo+= 30;
		fechasPago[i] = fechaTrabajo;
	}
}

//agregado
Fecha Prestamo::obtenerFechaAutorizacion()
{
	return fechaAutorizacion;
}
double Prestamo::obtenerValor()
{
	return valor;
}

void Prestamo::capturar()
{
	int nro = 0;
	persona sol;
	double vlr = 0;
	Fecha fa;
	// Estas dos variables sirven para decidir cual función se
	// utilizará para capturar los datos del solicitante
		int t = 0;
		cout << "\n----- Capturando los datos de un préstamo -----\n";
	cout << "Numero: ";
	cin >> nro;
	cout << "Tipo de captura de datos de solicitante (0=Abreviados, 1=Completo): ";
	cin >> t;
	cout << "Solicitante: ";
	// Se utiliza la función completa o abreviada, según la elección del usuario
	if (t==0)
	{	sol.capturar(true);
	}
	else
	{	sol.capturar();
	}
	// Se sigue pidiendo el valor del préstamo si el usuario insiste en ingresar un valor menor que cero
	while (vlr < 0)
	{
		cout << "Valor: ";
		cin >> vlr;
		if (vlr < 0)
		{
			cout << "Valor invalido, ingrese valor correcto" << endl;
		}
	}
	cout << "Fecha de Autorización (dd/mm/aaaa): ";
	cin >> fa;
	while (fa.obtenerDia() > 20)
	{
		cout << "Fecha de Autorizacion debe estar en los primeros 20 dias del mes" << endl;
		cout << "Intente de nuevo: ";
		cin >> fa;
	}
	establecer(nro, sol, vlr, fa);
}

void Prestamo::imprimir()
{
	cout << "\n----- Imprimiendo los datos de un prestamo -----\n";
	cout << "Numero de Prestamo: " << numero << "\n";
	cout << "Fecha de autorizacion: " << fechaAutorizacion << "\n";
	cout << "Fecha de entrega (tentativa): " << fechaEntrega << "\n";
	cout << "----- Detalle de Fechas de Pago -----\n";
	for (int i = 0; i < 6; i++)
	{	cout << "Cuota " << i << " : " << fechasPago[i] << endl;
```

```
        }
}

// Redefinición de función capturar de persona
void persona::capturar(bool x)
{
        char id [14];
        char pn [20];
        char pa [20];
        NumeroTelefonico tcasa;
        NumeroTelefonico tmov;
        cout << "\n------------- Datos Personales ------------" << endl;
        cout << "Identidad: ";
        cin.ignore();
        cin.getline(id, 14);
        cout << "Primer Nombre: ";
        cin.getline(pn, 20);
        cout << "Primer Apellido: ";
        cin.getline(pa, 20);
        cout << "-- Para numeros de telefono utilice este formato: (xxx) xxx-xxxx ext.xxx --\n" ;
        cout << "Telefono de Casa:\n";
        cin >> tcasa;
        cout << "Telefono Movil:\n";
        cin >> tmov;
        establecerIdentidad(id);
        establecerPrimerNombre(pn);
        establecerPrimerApellido(pa);
        establecerTelefonoCasa(tcasa);
        establecerTelefonoMovil(tmov);
}
```

RegistroPrestamos.cpp

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

bool validaFecha(Fecha, Fecha);

int main()
{
        int salir = 0;
        double montoTotal = 0;
        Fecha fechaLimite;
        double montoAcumulado = 0;

        cout << "Fecha limite para solicitud de prestamo (dd/mm/yyyy): ";
        cin >> fechaLimite;
        cout << "Monto total disponible para prestar: ";
        cin >> montoTotal;

        while (salir ==  0)
        {
                Fecha fa;
                Prestamo p;

                p.capturar();
                fa = p.obtenerFechaAutorizacion();
                if (validaFecha(fa, fechaLimite) == true)
                {
                        montoAcumulado+= p.obtenerValor();
                        p.imprimir();
                }
                if (montoAcumulado >= montoTotal)
                {
                        salir = 1;
                }
                else
                {
                        cout << "Desea salir (0=NO, 1=SI): ";
                        cin >> salir;
                }
        }

        system("PAUSE");
        return 0;
}

bool validaFecha(Fecha fa, Fecha fechaLimite)
{
        bool resultado = true;
        if (fa.obtenerYear() > fechaLimite.obtenerYear())
        {
                cout << "Fecha fuera de plazo" << endl;
                resultado = false;
        }
```

```
	else
	{
		if (fa.obtenerYear() == fechaLimite.obtenerYear())
		{
			if (fa.obtenerMes() > fechaLimite.obtenerMes())
			{
				cout << "Fecha fuera de plazo" << endl;
				resultado = false;
			}
			else
				if (fa.obtenerMes() == fechaLimite.obtenerMes())
				 {
					if (fa.obtenerDia() > fechaLimite.obtenerDia())
					{
						cout << "Fecha fuera de plazo" << endl;
						resultado = false;
					}
				}
		}
	}
	return resultado;
}
```

Otro ejercicio basado en el proyecto Préstamos

Planteamiento

Haciendo uso de la clase préstamo y sus clases relacionadas, trabajadas en el ejercicio “Proyecto Préstamos”, elabore un programa principal que cumpla con los siguientes requerimientos:

- Solicitar una fecha máxima para la autorización de préstamos.
- Solicitar un monto total disponible para prestar.
- Permitir que el usuario digite cuantos préstamos quiera, y para cada préstamo hacer lo siguiente:
- Si la fecha de autorización es mayor que la fecha máxima digitada al principio, solicitar al usuario que ingrese una nueva fecha, hasta que ingrese una fecha aceptable.
- Si el valor del préstamo es mayor que el monto total disponible, solicitar al usuario que ingrese un nuevo valor del préstamo, hasta que ingrese un valor aceptable.
- Si el valor del préstamo sobrepasa los Lps. 100,000 deben sumarse 3 días a la fecha tentativa de entrega.
- El valor del préstamo debe restarse al monto total disponible.
- El valor del préstamo debe acumularse para saber, al final, cuanto se prestó.
- Deben imprimirse los datos del préstamo.
- Debe preguntarse al usuario si desea salir del programa.
- Cuando el usuario decida salir del programa, debe imprimirse el monto total disponible, y el total prestado.

Diagrama de clases

Es idéntico al del primer ejemplo.

Código fuente

¡Adelante! Ya saben cómo hacerlo…

Último ejercicio basado en el proyecto Préstamos

Planteamiento

Se requiere un programa para la registro de préstamos en una cooperativa que debe reunir los siguientes requerimientos:

- Debe permitirse la captura de tantos préstamos como requiera el usuario.
- Al inicio del programa debe establecerse un valor máximo que puede tener cada préstamo.
- Si el valor del préstamo excede el valor máximo esperado debe mostrarse un mensaje de error.
- Si el valor del préstamo es menor a Lps. 5,000.00, debe preguntársele al usuario si desea "redondear" su préstamo a esa cantidad.
- Debe imprimir los datos completos del préstamo, incluyendo la fecha de entrega y las fechas de pago de las cuotas.
- Al final debe mostrarse el total prestado.

Diagrama de clases

Es idéntico al del primer ejemplo.

Código fuente

¡Adelante! Ya saben cómo hacerlo…