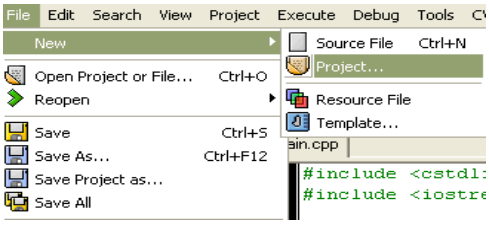
Tema 2: Práctica guiadas C+++

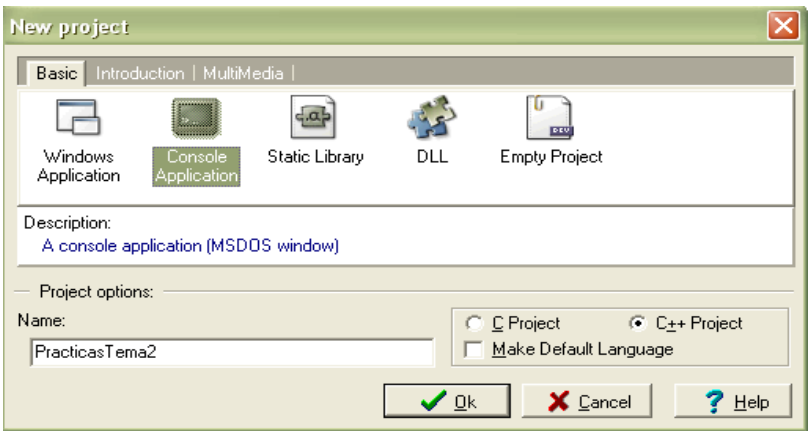
Nota: Este documento constituye material de apoyo a las prácticas obligatorias que posteriormente se deberán entregar.

Proceso de creación de un proyecto de programación en Dev C++

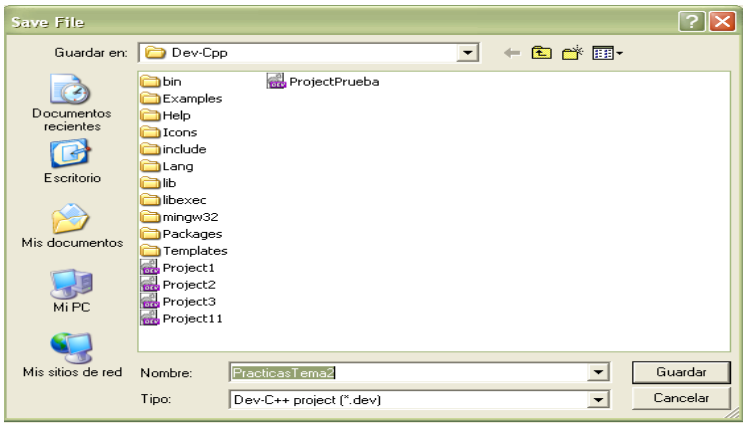
1. Crear nuevo proyecto Dev C++



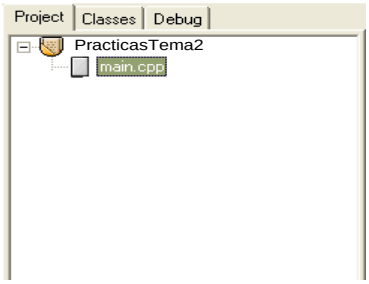
2. Seleccionar el tipo de aplicación C++ : **Aplicación de consola**



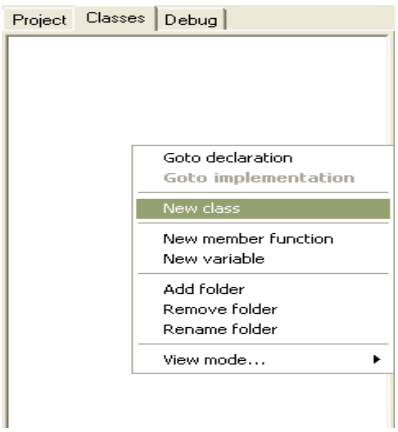
3. Indicamos ahora el directorio se va a guardar el archivo que describe el proyecto (*.dev)



4. Estructura organizativa creada por Dev C++ para nuestro proyecto



5. Crear nuevas clases en el proyecto



Práctica guiada 1.

Definimos un número racional como un decimal finito o infinito periódico (por ejemplo, el número decimal finito 0,75 es la representación decimal del número racional ¾

La representación en UML sería la siguiente:

Racional
-numerador : int -denominador : int
+visualizar() +Racional(entrada int num, entrada int den)

Clase racional.h

```
class Racional

{

    private:

            int numerador,denominador;

       public:
```

```
              Racional(int num,int den);

              void VisualizarRacional();

           ~Racional();

};
```

Racional.cpp

```
#include <cstdlib>

#include <iostream>

#include "racional.h"

using namespace std;

Racional::Racional(int num,int den)

{

        numerador = num;

    if (den == 0) den = 1; // el denominador no puede ser cero

    denominador = den;

}

void Racional::VisualizarRacional(){

cout<<numerador<<"/"<<denominador<<endl;
```

```
}

Racional::~Racional()

{

}
```

Nota: *observe como se imprime en pantalla con* ***cout . Para ello es necesario que importe en este archivo fuente lo siguiente***

#include <cstdlib>

#include <iostream>

using namespace std;

Utilice la clase Racional en el programa principal (*main.cpp*) y cree el numero racional (3/4) para después visualizarlo en pantalla.

Main.cpp

```
#include <cstdlib>

#include <iostream>

#include "racional.h"

using namespace std;

int main(int argc, char *argv[])

{

    Racional *r1=new Racional(3,4);

    r1->VisualizarRacional();

    system("PAUSE");

    return EXIT_SUCCESS;

}
```

Cuestiones:

1.-En el método visualizarRacional() existe una dependencia respecto al objeto cout el cual representa la salida estándar (la pantalla). Queremos desacoplar totalmente esta clase de su impresión por pantalla. Para ello añada un nuevo método a la clase que obtenga el valor decimal del número racional (tipo de dato float) representado e imprima este valor desde el programa principal por pantalla.

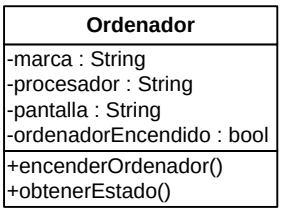
Práctica guiada 2

Represente mediante una clase un Ordenador (marca, procesador, pantalla, indicador de encendido o apagado) y diseñe además un método denominado *encenderOrdenador()* que imprimirá por pantalla el mensaje "Ordenador ya está encendido" si ya está encendido y si no es así cambiará el valor de la variable de estado para ponerlo en valor "encendido".

Diseñe otro método llamado *obtener*E*stado()* que imprima por pantalla el valor de cada uno de los atributos.

La clase debe tener un constructor con tres argumentos: marca, procesador y pantalla. El estado inicial del ordenador será de "apagado".

La representación en UML de esta clase sería:



ordenador.h

```
#include <string>

using namespace std;

class Ordenador

{

  private:

    string marca;

    string procesador;

    string pantalla;

    bool OrdenadorEncendido;

  public:

       Ordenador(string mar,string pro,string pan);

        void encenderOrdenador();

        void obtenerEstado();

               ~Ordenador();

};
```

Nota importante: Observa como se ha declarado una cadena en C++ con la clase **string**

```
    string marca;

    string procesador;

    string pantalla;
```

Para ello hemos importado la librería **<string.h>**

Ordenador.cpp

```
#include "ordenador.h"
#include <cstdlib>
#include <iostream>


using namespace std;
Ordenador::Ordenador(string mar,string pro,string pan)
{
   marca=mar;
   procesador=pro;
   pantalla=pan;
   OrdenadorEncendido=false;
}
void Ordenador::encenderOrdenador(){
     if (OrdenadorEncendido == true) // si está encendido...
      cout <<"El ordenador ya está encendido.";
    else // si no está encendido, encenderlo.
    {
      OrdenadorEncendido = true;
      cout <<"El ordenador se ha encendido.";
    }
}
void Ordenador::obtenerEstado(){
      cout<<"Estado del ordenador:" +marca + " Procesador :" +procesador +
       " Pantalla :" +pantalla<< endl;
    if (OrdenadorEncendido == true) // si el ordenador está encendido...
      cout<<"El ordenador está encendido.";
    else // si no está encendido...
      cout<<"El ordenador está apagado.";
  }
Ordenador::~Ordenador()
{
}
```

Nota importante: Observa cómo se concatena una cadena con el operador + en la siguiente instrucción

```
cout<<"Estado del ordenador:" +marca + " Procesador :" +procesador +

       " Pantalla :" +pantalla<< endl;
```

Cuestiones:

1.-Igual que el ejercicio anterior desacople totalmente la impresión por pantalla en la clase ordenador y realice esta tarea en la clase ProgramaPrincipal

Práctica guiada 3.

Representar un Cuenta Bancaría compuesta por un valor de saldo y un tipo de interés. La cuenta permitirá ingresar y retirar dinero, calcular los intereses de abono, cambiar el tipo de interés e imprimir el saldo actual de la cuenta.

A la hora de implementar esta clase hay que tener en cuenta que:

- Al establecer el tipo de interés se debe considerar que este no puede ser un valor negativo.
- La retirada de dinero sólo se puede hacer si hay saldo suficiente.
- Los intereses se calculan con la siguiente expresión
  `saldo += saldo * tipoDeInteres / 100;`

La representación en UML sería la siguiente:

CuentaBancaria
-saldo : double -tipo : double
+establecerTipo(entrada ti : double) +ingresarDinero(entrada cantidad : Double) +retirarDinero(entrada cantidad : double) +abonarIntereses() +imprimirSaldoActual() : String

Diseñe un programa principal que cree una cuenta bancaria con los siguientes movimientos:

- Ingreso de apertura:1.000.000
- Abono de intereses
- Ingreso :500.000
- Retirada:200.000
- Abono de intereses

Y que imprima el saldo resultado resultante.

`cuentabancaria.h`

```
#include <cstdlib>
#include <iostream>
using namespace std;
class CuentaBancaria
{
      private:
              double tipoDeInteres;
               double saldo;
       public:
               CuentaBancaria();
               void establecerTipoDeInteres(double ti);
               void ingresarDinero(double ingreso);
               void retirarDinero(double cantidad);
               double saldoActual();
               void abonarIntereses();
               ~CuentaBancaria();
};
```

cuentabancaria.cpp

```
#include "cuentabancaria.h" // class's header file


// class constructor
CuentaBancaria::CuentaBancaria(){
       saldo=0.0;
       tipoDeInteres=0.0;
}


 void CuentaBancaria::establecerTipoDeInteres(double ti)
  {
    if ( ti < 0)
    {
      cout<<"El tipo de interés no puede ser negativo";
      return; // retornar
    }
    tipoDeInteres = ti;
```

```
  }


 void CuentaBancaria::ingresarDinero(double ingreso)
  {
    saldo += ingreso;
  }


 void CuentaBancaria::retirarDinero(double cantidad)
  {
    if ( saldo - cantidad < 0)
    {
      cout<<"No tiene saldo suficiente";
      return;
    }
    // Hay saldo suficiente. Retirar la cantidad.
    saldo -= cantidad;
  }


 double CuentaBancaria::saldoActual()
  {
    return saldo;
  }


   void CuentaBancaria::abonarIntereses()
  {
    saldo += saldo * tipoDeInteres / 100;
  }
// class destructor
CuentaBancaria::~CuentaBancaria()
{
       // insert your code here
}
```

Main.cpp

```
#include <cstdlib>

#include <iostream>

#include "cuentabancaria.h"

using namespace std;


int main(int argc, char *argv[])

{

   // Abrir una cuenta con 1.000.000 a un 2%

    CuentaBancaria *Cuenta01 = new CuentaBancaria();

    Cuenta01->ingresarDinero(1000000);

    Cuenta01->establecerTipoDeInteres(2);


    cout.precision(7);

    cout<<Cuenta01->saldoActual()<<endl;

    Cuenta01->ingresarDinero(500000);

    Cuenta01->retirarDinero(200000);

    cout<<Cuenta01->saldoActual()<<endl;

    Cuenta01->abonarIntereses();

    cout<<Cuenta01->saldoActual()<<endl;

    system("PAUSE");

    return EXIT_SUCCESS;

}
```

Práctica guiada 3.

Cree un fichero de texto usando el objeto **ofstream** e imprima la cadena “Prueba de escritura en archivo” en él.

```
#include <cstdlib>

#include <iostream>

#include <fstream>


using namespace std;

int main(int argc, char *argv[])

{

   //llamada al constructor
```

```
   ofstream *archivo = new ofstream("c://fichero2.txt", ofstream::out);
    if (!(*archivo)){
          cout << "fallo en apertura" << endl;
          return -1;
    }
    (*archivo) << "prueba de escritura de fichero "; //escritura en el archivo de texto
   archivo->close(); //cierre del archivo
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Nota importante: Observe cómo se utiliza el objeto **ofstream.**

La escritura se realiza de igual manera que con **cout**

```
(*archivo) << "prueba de escritura de fichero ";
```