

2019-Practicas-POO-.pdf



Ant16L



Programación Orientada a Objetos



2º Grado en Ingeniería Informática



**Escuela Superior de Ingeniería
Universidad de Cádiz**

Practicas 2019 POO

Índice

Practicas.....	1
P0.....	2
Makefile.....	2
fecha.hpp	3
fecha.cpp.....	4
cadena.hpp	7
cadena.cpp.....	8
P1.....	11
Makefile.....	11
fecha.hpp	12
fecha.cpp.....	13
cadena.hpp	16
cadena.cpp.....	18
P2.....	21
articulo.hpp.....	21
articulo.cpp	22
tarjeta.hpp	23
tarjeta.cpp.....	25
usuario.hpp	28
usuario.cpp.....	30
P3.....	32
articulo.hpp.....	32
articulo.cpp	33
tarjeta.hpp	34
tarjeta.cpp.....	36
usuario.hpp	39
usuario.cpp.....	41
pedido.hpp	43
pedido.cpp	44
pedido-articulo.hpp.....	45
pedido-articulo.cpp	47
usuario-pedido.hpp	49
P4.....	50
articulo.hpp.....	50
articulo.cpp	52
pedido.hpp	54
pedido.cpp	55
tarjeta.hpp	56
tarjeta.cpp.....	58
usuario.hpp	61
usuario.cpp.....	63
pedido-articulo.hpp.....	65
pedido-articulo.cpp	67
usuario-pedido.hpp	69
usuario-pedido.cpp	69

Recomiendo leer antes también el PDF de consejos.

[PO](#)

[Inicio](#)

[Makefile](#)

```
EXES = test-P0-auto test-P0-consola
CXX = g++
CXXFLAGS = -g -Wall -std=c++11 -pedantic
CPPFLAGS = -DPO -I../Tests-auto -I.
VPATH = ../Tests-auto:.

CLASSES = cadena.o fecha.o
```

.PHONY: all clean

all: \$(EXES)

```
test-P0-auto: test-caso0-fecha-auto.o test-caso0-cadena-auto.o test-auto.o $(CLASSES)
$(CXX) $(CXXFLAGS) $^ -o $@
```

```
test-caso0-fecha-auto.o test-caso0-cadena-auto.o test-auto.o: \
    test-caso0-fecha-auto.cpp test-caso0-cadena-auto.cpp \
    test-auto.cpp test-auto.hpp fecha.hpp cadena.hpp
```

```
test-P0-consola: test-P0-consola.o $(CLASSES)
$(CXX) $(CXXFLAGS) $^ -o $@
```

clean:

```
@$(RM) $(EXES) *.o
```

```

fecha.hpp
#ifndef FECHA_HPP
#define FECHA_HPP

#include<iostream> //Para el 'operator<<'


class Fecha{
    private:
        ///VARIABLES///
        int dd_, mm_, aaaa_, anioActual_;

        ///FUNCIONES///
        void actualiza();
        bool comprueba() const;
        int dia_max() const;

    public:
        ///CONSTANTES///
        static const int AnnoMaximo = 2037;
        static const int AnnoMinimo = 1902;

        ///INVALIDA///
        class Invalida{
            private:
                const char *info_;
            public:
                Invalida(const char *in);
                const char* por_que() const;
        };
        ///CONSTRUCTOR///
        Fecha(int dd=0, int mm=0, int aaaa=0);
        Fecha(const char* c);
        Fecha(const Fecha &f); //constructor copia
        /*Constructor asignación default*/
        operator const char*() const; //conversor a const char*

        ///DESTRUCTOR///
        ~Fecha();

        ///OBSERVADORES///
        int dia() const;
        int mes() const;
        int anno() const;

        ///OPERADOR///
        Fecha operator+=(int i);           //OPERATOR SUMA-IGUAL          f+=i;
        Fecha operator-=(int i);           //OPERATOR RESTA-IGUAL          f-=i;
        Fecha operator++();               //OPERATOR PRE-INCREMENTO      ++f;
        Fecha& operator++(int);           //OPERATOR POST-INCREMENTO     f++;
        Fecha operator--();               //OPERATOR PRE-DECREMENTO      --f;
        Fecha& operator--(int);           //OPERATOR POST-DECREMENTO     f--;

    protected:
};

///OPERADORES ARITMÉTICOS///
Fecha operator+(Fecha f, int d);
Fecha operator-(Fecha f, int d);

///OPERADORES LÓGICOS///
bool operator==(const Fecha& f1, const Fecha& f2); //OPERATOR IGUAL QUE          ==
bool operator!=(const Fecha& f1, const Fecha& f2); //OPERATOR NO-IGUAL QUE         !=
bool operator>(const Fecha& f1, const Fecha& f2); //OPERATOR MAYOR QUE           >
bool operator<(const Fecha& f1, const Fecha& f2); //OPERATOR MENOR QUE           <
bool operator>=(const Fecha& f1, const Fecha& f2); //OPERATOR MAYOR O IGUAL QUE   >=
bool operator<=(const Fecha& f1, const Fecha& f2); //OPERATOR MENOR O IGUAL QUE   <=


///ENTRADA - SALIDA///
std::ostream& operator<<(std::ostream& os, const Fecha& f);
std::istream& operator>>(std::istream& is, Fecha& f);

#endif

```

```

fecha.cpp
#include <ctime>
#include <iostream>
#include "fecha.hpp"

using namespace std;

///FUNCIONES PRIVATE///
void Fecha::actualiza(){
try{
    while(mm_>12){
        mm_-=12;
        aaaa_++;
    }
    while (dd_ > dia_max()){
        dd_-= dia_max();
        if (++mm_ > 12){
            mm_= 1;
            aaaa_++;
        }
    }
    while (dd_ <= 0){
        if (--mm_ <=0){
            mm_= 12;
            aaaa_--;
        }
        dd_ += dia_max();
    }
    if(!comprueba()) {Fecha::Invalida in("Fecha invalida"); throw in;}
}
catch(Fecha::Invalida in) {cerr<<in.por_que()<<endl;}
}

bool Fecha::comprueba()const{
    return !(dd_>dia_max() || dd_<1 || mm_>12 || mm_<1 || aaaa_>Fecha::AnnoMaximo || aaaa_<Fecha::AnnoMinimo);
}

int Fecha::dia_max() const
{try{
    if(      mm_== 1 || mm_== 3 || mm_== 5 || mm_== 7 || mm_== 8 || mm_== 10 || mm_== 12 ) return 31;
    //el mes es de 31 días
    else if(mm_== 4 || mm_== 6 || mm_== 9 || mm_== 11 ) return 30;
    //el mes es de 30 días
    else if(mm_== 2 ){
        //¿AÑO BISIESTO?//
        if (aaaa_ % 4 == 0 && (aaaa_ % 400 == 0 || aaaa_ % 100 != 0)) return 29;
        //año bisiesto - febrero de 29 días
        else return 28; //año no bisiesto - febrero con 28 días
    }else {Fecha::Invalida in("Mes invalido"); throw in;}
}
catch (Fecha::Invalida in) {cerr << in.por_que() << endl;}
return 0;
}

///INVALIDA///
Fecha::Invalida::Invalida(const char *in): info_(in){}
inline const char* Fecha::por_que()const{return info_;}

///CONSTRUCTOR///
Fecha::Fecha(int dd, int mm, int aaaa)
:dd_(dd), mm_(mm), aaaa_(aaaa){
    std::time_t tiempo_calendario = std::time(0);
    std::tm* tiempo_descompuesto = std::localtime(&tiempo_calendario);

    if(aaaa_==0) aaaa_ = tiempo_descompuesto->tm_year + 1900;
    else while(aaaa_>9999) aaaa_-=10; //se toma solo los 4 primeros dígitos del año, si se introducen más se descartan
    if(mm_==0) mm_= tiempo_descompuesto->tm_mon + 1;
    if(dd_==0) dd_= tiempo_descompuesto->tm_mday;
}

```

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



```
        actualiza(); //lanza una ejeción si es invalida
    }
Fecha::Fecha(const char* c){
    sscanf(c, "%d/%d/%d", &dd_, &mm_, &aaaa_);
    std::time_t tiempo_calendario = std::time(0);
    std::tm* tiempo_descompuesto = std::localtime(&tiempo_calendario);
    if(aaaa_==0) aaaa_ = tiempo_descompuesto->tm_year + 1900;
        else while(aaaa_>9999) aaaa_-=10; //se toma solo los 4 primeros dígitos del año, si se introducen más se descartan
    if(mm_==0) mm_ = tiempo_descompuesto->tm_mon + 1;
    if(dd_==0) dd_ = tiempo_descompuesto->tm_mday;

    actualiza();//lanza una ejeción si es invalida
}
Fecha::Fecha(const Fecha &f):dd_(f.dia()), mm_(f.mes()), aaaa_(f.anno()){
    actualiza();//lanza una ejeción si es invalida
}

///DESTRUCTOR///
Fecha::~Fecha(){dd_=0;mm_=0;aaaa_=0;} //No es necesario, se puede dejar el por defecto

///OPERADOR///
Fecha Fecha::operator+=(int i){           //OPERATOR SUMA-IGUAL          f+=i;
    dd_+=i;
    actualiza();
    return *this;
}
Fecha Fecha::operator-=(int i){           //OPERATOR RESTA-IGUAL          f-=i;
    dd_-=i;
    actualiza();
    return *this;
}
Fecha Fecha::operator++(){               //OPERATOR PRE-INCREMENTO      ++f;
    *this+=1;
    actualiza();
    return *this;
}
Fecha& Fecha::operator++(int){           //OPERATOR POST-INCREMENTO     f++;
    Fecha *f= new Fecha(0);
    *f=*this;
    *this+=1;
    actualiza();
    return *f;
}
Fecha Fecha::operator--(){               //OPERATOR PRE-DECREMENTO      -f;
    *this-=1;
    actualiza();
    return *this;
}
Fecha& Fecha::operator--(int){           //OPERATOR POST-INCREMENTO     f--;
    Fecha *f= new Fecha(0);
    *f=*this;
    *this-=1;
    actualiza();
    return *f;
}

///OBSERVADOR///
inline int Fecha::dia() const {return dd_;}
inline int Fecha::mes() const {return mm_;}
inline int Fecha::anno() const {return aaaa_;}
```

```
///OPERADORES ARITMÉTICOS///
Fecha operator+(Fecha f, int d){           //OPERATOR SUMA
    Fecha t(f);
    t += d;
    return t;
}
```



WUOLAH

```

Fecha operator-(Fecha f, int d) {           //OPERATOR RESTA          ..f-d;
    Fecha t(f);
    t -= d;
    return t;
}

///OPERADORES LÓGICOS///
bool operator==(const Fecha& f1, const Fecha& f2) {           //OPERATOR IGUAL QUE      ==
    return (f1.dia()==f2.dia() && f1.mes()==f2.mes() && f1.anno()==f2.anno());
}
bool operator!=(const Fecha& f1, const Fecha& f2) {           //OPERATOR IGUAL QUE      !=
    return !(f1 == f2);
}
bool operator<(const Fecha& f1, const Fecha& f2) {           //OPERATOR MENOR QUE       <
    if(f1.anno() < f2.anno())      return true;
    else if (f1.anno() > f2.anno()) return false;
    else if (f1.mes() < f2.mes())   return true;
    else if (f1.mes() > f2.mes())   return false;
    else if (f1.dia() < f2.dia())   return true;
    else if (f1.dia() > f2.dia())   return false;
    else                           return false;
}
bool operator>(const Fecha& f1, const Fecha& f2) {           //OPERATOR MAYOR QUE       >
    return (f2 < f1);
}
bool operator>=(const Fecha& f1, const Fecha& f2) {           //OPERATOR MAYOR O IGUAL QUE >=
    return !(f1 < f2);
}
bool operator<=(const Fecha& f1, const Fecha& f2) {           //OPERATOR MENOR O IGUAL QUE <=
    return !(f2 < f1);
}

///ENTRADA - SALIDA///
std::ostream& operator<<(std::ostream& os, const Fecha& f){
    setlocale(LC_ALL, "");
    os << f.dia() << "/" << f.mes() << "/" << f.anno();
    return os;
}
std::istream& operator>>(std::istream& is, Fecha& f){
    setlocale(LC_ALL, "");
    char* s = new char[11];
    is>>s;
    Fecha f2(s);
    f=f2;
    delete[] s;
    is.ignore();

    return is;
}

///CONSTRUCTOR DE CONVERSIÓN///
Fecha::operator const char*() const{
    char *aux=new char[40];
    tm f= {};
    f.tm_mday=dd_;
    f.tm_mon =mm_-1;
    f.tm_year=aaaa_-1900;

    mktime(&f);
    strftime(aux, 40, "%A, %d de %B de %Y", &f);

    return aux;
}

```

```

cadena.hpp
#ifndef CADENA_HPP
#define CADENA_HPP

#include<iterator>

class Cadena{
    private:
        ///VARIABLES///
        char *s_;
        size_t tam_;

        //FUNCIONES PRIVADAS///
        void copiar(const Cadena& c);
        void copiar(const char*);

    public:
        //CONSTRUCTORES///
        Cadena(size_t l=0, char c=' ');
        Cadena(const Cadena& c);
        Cadena(const char* s);
        ~Cadena();           //destructor
        operator const char*() const; // EL OPERATOR CONST CHAR* SE CAMBIA POR LA FUNCION C_STR();

        //OBSERVADORES///
        const char* puts();
        const char* puts() const;           //devuelve una Cadena
                                            //const

        //FUNCIONES///
        size_t length();                  //devuelve el tamaño de la Cadena
        const size_t length() const;      //const
        size_t length(const char*c);     //devuelve el tamaño de la Cadena
        const size_t length(const char*s) const; //const
        const char* c_str() const;       //const

        //OPERADORES ARITMÉTICOS///
        Cadena operator+=(const Cadena& c); //concatena 2 cadenas y la copia en la primera

        //OPERADORES DE ASIGNACIÓN///
        Cadena& operator=(const Cadena& c);
        Cadena& operator=(const char* c); //versión: Cadena = Cadena;
                                         //versión: Cadena = const char*;

        //OPERADORES DE INDICE///
        char& operator[](size_t i); //NO comprueba si el índice es valido
        const char& operator[](size_t i) const; //const
        char& at(size_t i); //comprueba si el índice es valido
        const char& at(size_t i) const; //const
        Cadena substr(size_t i, size_t fin) const; //devuelve la Cadena comprendida entre i y fin

    protected:
};

//OPERADOR ARITMÉTICO///
Cadena operator+(const Cadena& c1, const Cadena& c2);

//OPERADORES LÓGICOS/// -compara el código ASCII de cada carácter/orden alfabético-
bool operator==(const Cadena& c1, const Cadena& c2);
bool operator!=(const Cadena& c1, const Cadena& c2);
bool operator>(const Cadena& c1, const Cadena& c2);
bool operator<(const Cadena& c1, const Cadena& c2);
bool operator>=(const Cadena& c1, const Cadena& c2);
bool operator<=(const Cadena& c1, const Cadena& c2);

#endif

```

```

cadena.cpp
#include "cadena.hpp"
#include <iostream>
#include <cstring>           //función strcat(), strcpy()....
#include <stdexcept>         //std::out_of_range
using namespace std;

//CONSTRUCTORES///
Cadena::Cadena(size_t l, char c):s_(new char[l+1]), tam_(l){
    while(l-->0) s_[l]=c;
    s_[tam_] = '\0';
}
Cadena::Cadena(const Cadena& c): s_{new char [c.tam_+1]}, tam_{c.tam_} {copiar(c);}
Cadena::Cadena(const char* s): s_{new char[length(s)+1]}, tam_{length(s)} {copiar(s);}
Cadena::~Cadena(){ //Destructor
    tam_= 0;
    delete[] s_;
}

//OBSERVADORES///
inline const char* Cadena::puts() {return s_;} //Devuelve la Cadena
inline const char* Cadena::puts() const {return s_;} //Version const

//FUNCIONES///
size_t Cadena::length() {return tam_;}           //Devuelve el tamaño de una Cadena
const size_t Cadena::length() const {return tam_;} //const
size_t Cadena::length(const char*s){
    int i=0;
    while(s[i]) i++;
    return i;
}
const size_t Cadena::length(const char*s) const{
    int i=0;
    while(s[i]) i++;
    return i;
}

//OPERADORES ARITMÉTICOS///
Cadena Cadena::operator+=(const Cadena& c){
    char* caux=new char[tam_+1];
    strcpy(caux, s_);
    tam_+= c.tam_;
    delete[] s_;
    s_ = new char[tam_+1];
    int i = 0;
    while(caux[i]){
        s_[i] = caux[i];
        i++;
    }
    int j = i;
    i = 0;
    while(c[i]){
        s_[j] = c.s_[i];
        i++; j++;
    }
    s_[j] = '\0';
    return *this;
}

//OPERADORES DE ASIGNACIÓN///
Cadena& Cadena::operator=(const Cadena& c){ //versión: Cadena = Cadena;
    if ( *this != c ) copiar(c);
    return *this;
}

```

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



```
Cadena& Cadena::operator= (const char* c){           //versión: Cadena = const char*;
    Cadena cc(c);
    if (*this != cc) copiar(c);
    return *this;
}

///OPERADORES DE INDICE///
char& Cadena::operator[](size_t i){return s_[i];}           //NO comprueba si el indice es valido
const char& Cadena::operator[](size_t i) const{return s_[i];} //const
char& Cadena::at(size_t i){                                //Comprueba si el índice está fuera de rango
    if (i < tam_) return s_[i]; //No se comprueba i >= 0 porque i es unsigned int, nunca puede ser menos que 0
    else{
        throw std::out_of_range ("Funcion at(): indice fuera de rango de la Cadena");
    }
}
const char& Cadena::at(size_t i) const{                     //const
    if (i < tam_) return s_[i]; //No se comprueba i >= 0 porque i es unsigned int, nunca puede ser menos que 0
    else{
        throw std::out_of_range e("Funcion at(): indice fuera de rango de la Cadena");
    }
}
Cadena Cadena::substr(size_t i, size_t fin) const{         //devuelve la Cadena comprendida entre i y final
    if (i > tam_ || fin > tam_){                           //si los indices excede algun extremo se lanza una excepción
        throw std::out_of_range e("Función substr(): índices fuera de rango");
    }
    if (i > fin) std::swap(i, fin);                         //Si los índices se cruzan se cambian
    Cadena caux(fin - i + 1);                               //nueva cadena con el tamaño de la diferencia +1
    int j = 0;
    while (i <= fin){
        caux[j] = s_[i];
        i++; j++;
    }
    caux[j] = '\0';
    return caux;
}

///FUNCIONES PRIVADAS///
void Cadena::copiar(const Cadena& c){ //Recibe una Cadena y la copia en otra
    tam_ = c.tam_;
    delete[] s_;
    s_ = new char[tam_ + 1];
    strcpy(s_, c.s_);
}
void Cadena::copiar(const char* c){ //Recibe una Cadena de bajo nivel y la copia a una Cadena
    tam_ = strlen(c);
    delete[] s_;
    s_ = new char[tam_ + 1];
    strcpy(s_, c);
}

///OPERADOR ARITMÉTICO///
Cadena operator+(const Cadena& c1, const Cadena& c2){
    Cadena caux(c1);
    return caux += c2;
}

///OPERADORES LÓGICOS/// -compara el código ASCII de cada carácter/orden alfabético-
bool operator==(const Cadena& c1, const Cadena& c2){
    if(c1.length()!=c2.length()) return false;

    int last = std::min(c1.length(), c2.length());

    for(int i = 0; i < last; i++)      if (c1.at(i) != c2.at(i))      return false;
    return true;
}
```



WUOLAH

```

bool operator!=(const Cadena& c1, const Cadena& c2){return !(c1==c2);}
bool operator<(const Cadena& c1, const Cadena& c2){
    if(c1.length()>c2.length()) return false;
    if(c1.length()<c2.length()) return true;
    int last = std::min(c1.length(), c2.length());
    for(int i = 0; i < last; i++)
        if (static_cast<int>(c1.at(i)) > static_cast<int>(c2.at(i))) return false;
    return true;
}
bool operator>(const Cadena& c1, const Cadena& c2){return (c2<c1);}
bool operator>=(const Cadena& c1, const Cadena& c2){
    if(c1==c2) return true;
    return (c2<c1);}
bool operator<=(const Cadena& c1, const Cadena& c2){
    if(c1==c2) return true;
    return (c1<c2);}

Cadena::operator const char*() const{return s_;} // EL OPERATOR CONST CHAR* SE CAMBIA POR LA FUNCION C_STR();

```

P1

Mejora de la P0

Makefile

```
EXES = test-P1-auto test-P1-consola  
CXX = g++  
CXXFLAGS = -g -Wall -std=c++14 -pedantic  
CPPFLAGS = -DP1 -I../Tests-auto -I.  
VPATH = ../Tests-auto:  
  
CLASSES = cadena.o fecha.o
```

```
.PHONY: all clean
```

```
all: $(EXES)
```

```
test-P1-auto: test-caso0-fecha-auto.o test-caso0-cadena-auto.o test-auto.o $(CLASSES)  
$(CXX) $(CXXFLAGS) $^ -o $@
```

```
test-caso0-fecha-auto.o test-caso0-cadena-auto.o test-auto.o: \  
    test-caso0-fecha-auto.cpp test-caso0-cadena-auto.cpp \  
    test-auto.cpp test-auto.hpp fecha.hpp cadena.hpp
```

```
test-P1-consola: test-P1-consola.o $(CLASSES)  
$(CXX) $(CXXFLAGS) $^ -o $@
```

```
clean:
```

```
@$(RM) $(EXES) *.o
```

```

fecha.hpp
#ifndef FECHA_HPP
#define FECHA_HPP

#include<iostream> //Para el 'operator<<' 
#include <iomanip>
#include <cstdio>
#include <ctime>

class Fecha{
private:
    ////VARIABLES////
    int dd_, mm_, aaaa_, anioActual_;

    ////FUNCIONES////
    void actualiza();
    bool comprueba() const;
    int dia_max() const;

public:
    ////CONSTANTES////
    static const int AnnoMaximo = 2037;
    static const int AnnoMinimo = 1902;

    ////INVALIDA////
    class Invalida{
        private:
            const char *info_;
        public:
            Invalida(const char *in);
            const char* por_que() const;
    };

    ////CONSTRUCTOR////
    explicit Fecha(int dd=0, int mm=0, int aaaa=0);
    Fecha(const char* c);
    /*Constructor asignación default*/
    const char* cadena() const; //conversor a const char*

    ////OBSERVADOR////
    const int dia() const noexcept{return dd_;}
    const int mes() const noexcept{return mm_;}
    const int anno() const noexcept{return aaaa_;}

    ////OPERADOR////
    Fecha& operator+=(int i);           //OPERATOR SUMA-IGUAL          f+=i;
    Fecha& operator-=(int i);           //OPERATOR RESTA-IGUAL          f-=i;
    Fecha& operator++();               //OPERATOR PRE-INCREMENTO      ++f;
    Fecha operator++(int);             //OPERATOR POST-INCREMENTO     f++;
    Fecha& operator--();               //OPERATOR PRE-DECREMENTO      --f;
    Fecha operator--(int);             //OPERATOR POST-DECREMENTO     f--;
    Fecha operator+( int d)const;
    Fecha operator-( int d)const;

protected:
};

////OPERADORES LÓGICOS////
bool operator==(const Fecha& f1, const Fecha& f2)noexcept;           //OPERATOR IGUAL QUE          ==
bool operator!=(const Fecha& f1, const Fecha& f2)noexcept;           //OPERATOR NO-IGUAL QUE        !=
bool operator>(const Fecha& f1, const Fecha& f2)noexcept;            //OPERATOR MAYOR QUE          >
bool operator<(const Fecha& f1, const Fecha& f2)noexcept;            //OPERATOR MENOR QUE          <
bool operator>=(const Fecha& f1, const Fecha& f2)noexcept;           //OPERATOR MAYOR O IGUAL QUE   >=
bool operator<=(const Fecha& f1, const Fecha& f2)noexcept;           //OPERATOR MENOR O IGUAL QUE   <=


////ENTRADA - SALIDA////
std::ostream& operator<<(std::ostream& os, const Fecha& f)noexcept;
std::istream& operator>>(std::istream& is, Fecha& f);

#endif

```

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



fecha.cpp

```
#include "fecha.hpp"
#include <new>

using namespace std;

///FUNCIONES PRIVATE///
void Fecha::actualiza(){
    while(mm_>12){
        mm_-=12;
        aaaa_++;
    }
    while (dd_ > dia_max()){
        dd_-= dia_max();
        if (++mm_ > 12){
            mm_= 1;
            aaaa_++;
        }
    }
    while (dd_ <= 0){
        if (--mm_ <=0){
            mm_= 12;
            aaaa_--;
        }
        dd_+= dia_max()
    }
    if(!comprueba()) {throw Fecha::Invalida ("Fecha invalida");}
}
bool Fecha::comprueba()const{
    return !(dd_ > dia_max() || dd_ < 1 || mm_ > 12 || mm_ < 1 || aaaa_ > Fecha::AnnoMaximo || aaaa_ < Fecha::AnnoMinimo );
}
int Fecha::dia_max() const
{
    if(      mm_ == 1 || mm_ == 3 || mm_ == 5 || mm_ == 7 || mm_ == 8 || mm_ == 10 || mm_ == 12 ) return 31;
    //el mes es de 31 días
    else if(mm_ == 4 || mm_ == 6 || mm_ == 9 || mm_ == 11 ) return 30;
    //el mes es de 30 días
    else if(mm_ == 2){
        //AÑO BISIESTO?/
        if (aaaa_ % 4 == 0 && (aaaa_ % 400 == 0 || aaaa_ % 100 != 0)) return 29; //año bisiesto - febrero de 29 días
        else return 28; //año no bisiesto - febrero con 28 días
    }else {throw Fecha::Invalida ("Mes invalido");}
    return 0;
}

///INVALIDA///
Fecha::Invalida::Invalida(const char *in): info_(in){}
const char* Fecha::Invalida::por_que()const{return info_;}

///CONSTRUCTOR///
Fecha::Fecha(int dd, int mm, int aaaa)
:dd_(dd), mm_(mm), aaaa_(aaaa){
    std::time_t tiempo_calendario = std::time(0);
    std::tm* tiempo_descompuesto = std::localtime(&tiempo_calendario);

    if(aaaa_==0) aaaa_= tiempo_descompuesto->tm_year + 1900;
    else while(aaaa_>9999) aaaa_-=10; //se toma solo los 4 primeros dígitos del año, si se introducen más se descartan
    if(mm_==0) mm_= tiempo_descompuesto->tm_mon + 1;
    if(dd_==0) dd_= tiempo_descompuesto->tm_mday;

    if(!comprueba()) throw Fecha::Invalida ("Fecha invalida");
}
```



WUOLAH

```

Fecha::Fecha(const char* c){
    dd_=-2; mm_=-2;aaaa_=-2;
    sscanf(c, "%d/%d/%d", &dd_, &mm_, &aaaa_);

    std::time_t tiempo_calendario = std::time(0);
    std::tm* tiempo_descompuesto = std::localtime(&tiempo_calendario);
    if(aaaa_==0) aaaa_ = tiempo_descompuesto->tm_year + 1900;
        else while(aaaa_>9999) aaaa_-=10; //se toma solo los 4 primeros dígitos del año, si se introducen más se descartan
    if(mm_==0) mm_ = tiempo_descompuesto->tm_mon + 1;
    if(dd_==0) dd_ = tiempo_descompuesto->tm_mday;

    if(!comprueba()) throw Fecha::Invalida ("Fecha invalida");
}

```

```

////OPERADOR////
Fecha& Fecha::operator+=(int i){           //OPERATOR SUMA-IGUAL          f+=i;
    this->dd_+=i;
    actualiza();

    return *this;
}
Fecha& Fecha::operator-=(int i){           //OPERATOR RESTA-IGUAL          f-=i;
    *this +=(-i);
    return *this;
}
Fecha& Fecha::operator++(){                //OPERATOR PRE-INCREMENTO      ++f;
    *this +=(1);
    return *this;
}
Fecha Fecha::operator++(int){               //OPERATOR POST-INCREMENTO     f++;
    Fecha t(*this);
    *this +=(1);
    return t;
}
Fecha& Fecha::operator--(){                //OPERATOR PRE-DECREMENTO      --f;
    *this +=(-1);
    return *this;
}
Fecha Fecha::operator--(int){               //OPERATOR POST-INCREMENTO     f--;
    Fecha t(*this);
    *this +=(-1);
    return t;
}

```

```

////OPERADORES ARITMÉTICOS////
Fecha Fecha::operator+( int d ) const{       //OPERATOR SUMA                  ..f+d;
    Fecha t(*this);
    t +=(d);
    return t;
}
Fecha Fecha::operator-(int d) const {         //OPERATOR RESTA                 ..f-d;
    Fecha t(*this);
    t +=(-d);
    return t;
}

```

```

////OPERADORES LÓGICOS////
bool operator==(const Fecha& f1, const Fecha& f2)noexcept { //OPERATOR IGUAL QUE          ==
    return (f1.dia()==f2.dia() && f1.mes()==f2.mes() && f1.anno()==f2.anno());
}
bool operator!=(const Fecha& f1, const Fecha& f2)noexcept { //OPERATOR IGUAL QUE          !=
    return !( f1 == f2 );
}
bool operator<(const Fecha& f1, const Fecha& f2)noexcept { //OPERATOR MENOR QUE          <
    if      (f1.anno() < f2.anno())   return true;
    else if (f1.anno() > f2.anno())   return false;
}

```

```

else if (f1.mes() < f2.mes()) return true;
else if (f1.mes() > f2.mes()) return false;
else if (f1.dia() < f2.dia()) return true;
else if (f1.dia() > f2.dia()) return false;
else return false;
}
bool operator >(const Fecha& f1, const Fecha& f2)noexcept { //OPERATOR MAYOR QUE >
    return (f2 < f1);
}
bool operator>=(const Fecha& f1, const Fecha& f2)noexcept { //OPERATOR MAYOR O IGUAL QUE >=
    return !(f1 < f2);
}
bool operator<=(const Fecha& f1, const Fecha& f2)noexcept { //OPERATOR MENOR O IGUAL QUE <=
    return !(f2 < f1);
}

```

///ENTRADA - SALIDA///

```

std::ostream& operator<<(std::ostream& os, const Fecha& f) noexcept{
    os<<f.cadena();
    return os;
}
std::istream& operator>>(std::istream& is, Fecha& f){
    char *cadAux = new char[11];
    is.width(11);    //Se limita el número de caracteres leidos al formato "dd/mm/aaaa\0"
    is >> cadAux;
    try {
        f = cadAux;
    }catch ( Fecha::Invalida e ){
        is.setstate(std::ios::failbit);    //Se marca failbit como activo
        delete[] cadAux;
        throw Fecha::Invalida("Error: operador >>: error de formato o al construir el objeto. Formato: dd/mm/aaaa");
    }
    delete[] cadAux;
    return is;
}

```

///CONSTRUCTOR DE CONVERSIÓN///

```

const char* Fecha::cadena() const{

    static char* buffer = new char[40];
    tm f= {};
    f.tm_mday=dd_;
    f.tm_mon=mm_-1;
    f.tm_year=aaaa_-1900;

    mktime(&f);
    strftime(buffer, 40, "%A %d de %B de %Y", &f);

    return buffer;
}

```

```

cadena.hpp
#ifndef CADENA_HPP
#define CADENA_HPP

#include <functional>
#include <iterator>           //los iteradores
#include <iostream>
#include <cstring>            //función strcat(), strcpy()....
#include <stdexcept>          //std::out_of_range

class Cadena{
    private:
        /////VARIABLES////
        size_t tam_;
        char *s_;

        /////FUNCIONES PRIVADAS////
        void copiar(const Cadena& c);
        void copiar(const char*);

    public:
        /////CONSTRUCTORES////
        explicit Cadena(size_t l=0, char c=' ');
        Cadena(const char* s);
        Cadena(const Cadena& c);
        Cadena(Cadena&& c); //movimiento
        ~Cadena();           //destructor

        /////OBSERVADORES////
        const char* puts()      {return s_}; //Devuelve la Cadena
        const char* puts() const {return s_}; //Version const

        /////FUNCIONES////
        size_t length() noexcept{return tam_}; }
        const size_t length() const noexcept{return tam_}; }
        size_t length(const char*c) noexcept;
        const size_t length(const char*s) const noexcept;
        const char* c_str() const noexcept;

        /////OPERADORES ARITMÉTICOS////
        Cadena operator+=(const Cadena& c); //concatena 2 cadenas y la copia en la primera

        /////OPERADORES DE ASIGNACIÓN////
        Cadena& operator=(const Cadena& c); //versión: Cadena = Cadena;
        Cadena& operator=(const char* c); //versión: Cadena = const char*;
        Cadena& operator=(Cadena&&c);

        /////OPERADORES DE INDICE////
        char& operator[](size_t i) noexcept; //NO comprueba si el indice es valido
        const char& operator[](size_t i) const noexcept; //NO comprueba si el indice es valido, const
        char& at(size_t i); //comprueba si el indice es valido
        const char& at(size_t i) const; //comprueba si el indice es valido, const
        Cadena substr(size_t i, size_t fin) const; //devuelve la Cadena comprendida entre i y final

        /////ITERADORES////
        typedef char* iterator;
        typedef const char* const_iterator;
        typedef std::reverse_iterator<iterator> reverse_iterator;
        typedef std::reverse_iterator<const_iterator> const_reverse_iterator;

        /////FUNCIONES CON ITERADORES////
        iterator begin() noexcept {return s_}; //funcion begin();
        const_iterator begin() const noexcept {return const_iterator(s_)}; //funcion begin() const;
        const_iterator cbegin() const noexcept {return const_iterator(s_)}; //funcion cbegin() const;
        reverse_iterator rbegin() noexcept {return reverse_iterator(end())}; //funcion begin();
        const_reverse_iterator rbegin() const noexcept {return const_reverse_iterator(end())}; //funcion rbegin() const;
        const_reverse_iterator crbegin() const noexcept {return const_reverse_iterator(end())}; //funcion crbegin() const;
        iterator end() noexcept {return s_ + tam_}; //funcion end();
        const_iterator end() const noexcept {return const_iterator(s_ + tam_)}; //funcion end() const;
        const_iterator cend() const noexcept {return const_iterator(s_ + tam_)}; //funcion cend() const;

```

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



```
reverse_iterator rend() noexcept {return reverse_iterator(begin());}
const_reverse_iterator rend() const noexcept {return const_reverse_iterator(begin());}
const_reverse_iterator rend() const noexcept {return const_reverse_iterator(begin());}

protected:
};

<:///OPERADOR ARITMÉTICO///>
Cadena operator+(const Cadena& c1, const Cadena& c2);

<:///OPERADORES LÓGICOS///> -compara el código ASCII de cada carácter/orden alfabético-
bool operator==(const Cadena& c1, const Cadena& c2);
bool operator!=(const Cadena& c1, const Cadena& c2);
bool operator>(const Cadena& c1, const Cadena& c2);
bool operator<(const Cadena& c1, const Cadena& c2);
bool operator>=(const Cadena& c1, const Cadena& c2);
bool operator<=(const Cadena& c1, const Cadena& c2);

<:///FLUJO ENTRADA - SALIDA///>
std::ostream& operator<<(std::ostream& os, const Cadena& c) noexcept;
std::istream& operator>>(std::istream& is, Cadena& c)noexcept;

/*ESTO NO ES DE LA P1, SINO DE LA P2 EN ADELANTE, TE LO DA AL FINAL DEL ENUNCIADO*/
// Para P2 y ss.
// Especialización de la plantilla hash<T>para definir la
// función hash a utilizar con contenedores desordenados de
// Cadena, unordered_[set|map|multiset|multimap].
```

```
namespace std{
    template<>struct hash <Cadena> {
        size_t operator()(const Cadena& cad) const {
            // conversión const char* ->string
            return hash <string> {}(cad.c_str());
        }
    };
}
```



```

cadena.cpp
#include "cadena.hpp"
using namespace std;

<:///CONSTRUCTORES///
Cadena::Cadena(size_t l, char c):tam_(l), s_(new char[l+1]) {
    while(l-->0) s_[l]=c;
    s_[tam_] = '\0';
}
Cadena::Cadena(const char* s): tam_{length(s)}, s_{new char[length(s)+1]}\{copiar(s);}
Cadena::~Cadena() //Destructor
{
    tam_ = 0;
    delete[] s_;
}
Cadena::Cadena(const Cadena& c):tam_{c.tam_}, s_{new char [c.tam_+1]} \{copiar(c);\}
Cadena::Cadena(Cadena&& c): tam_{c.tam_}, s_{c.s_} {
    c.tam_ = 0;
    c.s_=nullptr;
}



```

```

Cadena& Cadena::operator=(Cadena&&c){ //MOVIMIENTO
    tam_=c.tam_;
    delete[] s_;
    s_=c.s_;
    c.tam_=0;
    c.s_=nullptr;
    return *this;
}

////OPERADORES DE INDICE////
char& Cadena::operator[](size_t i) noexcept {return s_[i];} //NO comprueba si el indice es valido
const char& Cadena::operator[](size_t i) const noexcept{return s_[i];} //NO comprueba si el indice es valido, const
char& Cadena::at(size_t i){ //Comprueba si el índice está fuera de rango
    if (i < tam_) return s_[i]; //No se comprueba i >= 0 porque i es unsigned int, nunca puede ser menos que 0
    else throw std::out_of_range ("Funcion at(): indice fuera de rango de la Cadena");
}
const char& Cadena::at(size_t i) const{ //Versión const
    if (i < tam_) return s_[i]; //No se comprueba i >= 0 porque i es unsigned int, nunca puede ser menos que 0
    else throw std::out_of_range ("Funcion at(): indice fuera de rango de la Cadena");
}
Cadena Cadena::substr(size_t i, size_t tam) const{ //devuelve la Cadena comprendida desde i con tamaño t
    if ( i > tam_ || tam > tam_ || (i + tam) > tam_) //Si los índices están fuera de rango se lanza una excepción
        throw std::out_of_range ("Funcion substr(): Indices fuera de rango");

    char *cadAux = new char[tam + 1]; //Uno más por el terminador

    int j = 0;
    unsigned int fin = i + tam;
    while ( i < fin ){
        cadAux[j] = s_[i];
        ++i;
        ++j;
    }
    cadAux[j] = '\0';
    Cadena cc(cadAux);
    delete[] cadAux;
    return cc;
}

////FUNCIONES PRIVADAS////
void Cadena::copiar(const Cadena& c){ //Recibe una Cadena y la copia en otra
    tam_=c.tam_;
    delete[] s_;
    s_= new char[tam_+1];
    strcpy(s_, c.s_);
}

void Cadena::copiar(const char* c){ //Recibe una Cadena de bajo nivel y la copia a una Cadena
    tam_=strlen(c);
    delete[] s_;
    s_= new char[tam_+1];
    strcpy(s_, c);
}

////OPERADOR ARITMÉTICO////
Cadena operator+(const Cadena& c1, const Cadena& c2){
    Cadena caux(c1);
    return caux += c2;
}

////OPERADORES LÓGICOS/// -compara el código ASCII de cada carácter/orden alfabético-
bool operator==(const Cadena& c1, const Cadena& c2){
    if(c1.length()!=c2.length()) return false;
    int last = std::min(c1.length(), c2.length());
    for(int i = 0; i < last; i++)
        if (c1.at(i) != c2.at(i)) return false;
    return true;
}

```

```

bool operator!=(const Cadena& c1, const Cadena& c2){return !(c1==c2);}
bool operator<(const Cadena& c1, const Cadena& c2){
    if(c1.length()>c2.length() ) return false;
    if(c1.length()<c2.length()) return true;
    int last = c1.length();
    for(int i = 0; i < last; i++)
        if (static_cast<int>(c1.at(i)) < static_cast<int>(c2.at(i))) return true;
    return false;
}
bool operator>(const Cadena& c1, const Cadena& c2){return (c2<c1);}
bool operator>=(const Cadena& c1, const Cadena& c2){
    if(c1==c2) return true;
    return (c2<c1);}
bool operator<=(const Cadena& c1, const Cadena& c2){
    if(c1==c2) return true;
    else return (c1<c2);}

////FLUJO ENTRADA - SALIDA////
std::ostream& operator<<(std::ostream& os, const Cadena& c) noexcept{
    os<<c.c_str();
    return os;
}
std::istream& operator>>(std::istream& is, Cadena& c)noexcept{
    char s[33] = "";
    is.width(33);
    is>>s;
    c = s;
    return is;
}

```

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



P2

3 clases nuevas

```
articulo.hpp
#ifndef ARTICULO_HPP
#define ARTICULO_HPP

#include "fecha.hpp"
#include "cadena.hpp"

class Articulo{
    private:
        Cadena referencia_;
        Cadena titulo_;
        Fecha f_publi_;
        double precio_;
        unsigned stock_;
    public:
        ///CONSTRUCTOR///
        Articulo(const Cadena& r, const Cadena& t, const Fecha& f, double p, unsigned s);

        ///OBSERVADORES///
        Cadena referencia() const;
        Cadena titulo() const;
        Fecha f_publi() const;
        double precio() const;
        unsigned stock() const;
        double& precio();
        unsigned& stock();
    protected:
};
std::ostream& operator <<(std::ostream& os, const Articulo& ar)noexcept;

#endif
```



```

articulo.cpp
#include "articulo.hpp"
#include "fecha.hpp"
#include "cadena.hpp"
#include <iomanip>
#include <stdlib.h>
#include <iostream>
#include <locale>

using namespace std;

///CONSTRUCTOR///
Articulo::Articulo(const Cadena& r, const Cadena& t, const Fecha& f, double p, unsigned s)
:referencia_(r), titulo_(t), f_publi_(f), precio_(p), stock_(s){}

///OBSERVADORES///
Cadena Articulo::referencia()const {return referencia_;}
Cadena Articulo::titulo() const {return titulo_;}
Fecha Articulo::f_publi() const {return f_publi_;}
double Articulo::precio() const {return precio_;}
unsigned Articulo::stock() const {return stock_;}
double& Articulo::precio() {return precio_;}
unsigned&Articulo::stock() {return stock_;}

std::ostream& operator <<(std::ostream& os, const Articulo& ar)noexcept{
    std::locale::global( std::locale( "" ) );
    os<<"["<<ar.referencia()<<"]\"";
    os<<ar.titulo()<<"\", "<<ar.f_publi().anno()<<". "<<fixed<<setprecision(2)<<ar.precio()<< €.";
    return os;
}

```

```

tarjeta.hpp
#ifndef TARJETA_HPP
#define TARJETA_HPP
#include "cadena.hpp"
#include "tarjeta.hpp"
#include "articulo.hpp"
#include "usuario.hpp"

class Usuario;
class Clave;
///////////////////////////////
///CLASE *NUMERO*///-----
//-----
class Numero{
    private:
        ///VARIABLES///
        Cadena numero_;

    public:
        ///CONSTANTES///
        enum Razon {LONGITUD, DIGITOS, NO_VALIDO};

        ///CONSTRUCTORES///
        Numero(const Cadena&);
        ~Numero();//destructor

        ///OPERADORES///
        operator const char*()const;

        ///OBSERVADORES///
        const Cadena& num() const{return numero_}

        ///INCORRECTO///
        class Incorrecto{
            public:
                Incorrecto(const Numero::Razon r);
                Numero::Razon razon()const;
            private:
                Numero::Razon r_;
        };
};

void eliminarChar(Cadena& cad, size_t pos);

///////////////////////////////
///CLASE *TARJETA*///-----
//-
class Tarjeta{
    public:
        ///CONSTANTES///
        enum Tipo {Otro, VISA, Mastercard, Maestro, JCB, AmericanExpress};

        ///CONSTRUCTORES///
        Tarjeta(const Numero& n, Usuario& usuario, const Fecha& caducidad);
        Tarjeta(const Tarjeta& that) = delete;
        Tarjeta& operator=(const Tarjeta &)=delete;
        ~Tarjeta();

        ///OBSERVADORES///
        const Tarjeta::Tipo& tipo() const{return tipo_;}
        const Numero& numero() const{return numero_;}
        Usuario* titular() const{return usuario_;}
        const Fecha& caducidad() const{return caducidad_;}
        const bool activa()const{return activa_;}

        bool &activa(bool a=true);
};

```

```

////CADUCADA////
class Caducada{
public:
    Caducada(const Fecha& f);
    const Fecha& cuando()const{return fecha_caducada;}
private:
    Fecha fecha_caducada;
};

////NUM_DUPPLICADO////
class Num_duplicado{
public:
    Num_duplicado(const Numero& );
    const Numero& que()const;
private:
    Numero numer_;
};

////DESACTIVADA////
class Desactivada{
public:
    Desactivada();
private:
};

////FUNCIONES////
void anula_titular();
Tipo esTipo();

friend std::ostream& operator<<(std::ostream& os, const Tarjeta& t);

private:
    ////VARIABLES////
    Numero numero_;           //numero troquelado
    Usuario* const usuario_;   //su titular
    Fecha caducidad_;         //caducidad
    bool activa_;              //1 == activa ; 0==no
    Tipo tipo_;                //tipo de la tarjeta
};

std::ostream& operator <<(std::ostream& os, const Tarjeta& t);
std::ostream& operator <<(std::ostream& os, const Tarjeta::Tipo& tipo);
bool operator <(const Numero& a, const Numero& b);
bool operator <(const Tarjeta& a, const Tarjeta& b);
bool operator >(const Tarjeta& a, const Tarjeta& b);
#endif

```

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



```
tarjeta.cpp
#include "tarjeta.hpp"
#include <iomanip>
#include <cstring>
#include <string.h>
#include <cctype>
using namespace std;
//algoritmo de luhn
bool luhn(const Cadena&);

///////////////////////////////for (auto const& i : u.compra())
////CLASE *NUMERO*///-----
//-----
///CONSTRUCTORES///
Numero::Numero(const Cadena& num){

    char keys[] = "./ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
    size_t ii = strcspn (num.c_str(), keys);
    if(ii<num.length()) throw Incorrecto(Razon::DIGITOS);
    char * pch;
    char *aux = new char[30];
    pch = strpbrk (const_cast<char*>(num.c_str()), "1234567890");
    int i=0;
    while (pch != NULL){
        aux[i++]=*pch;
        pch = strpbrk (pch+1,"1234567890");
    }
    aux[i]='\0';
    Cadena n(aux);
    delete[] aux;

    if(n.length()>19 || n.length()<13) //si no esta entre 13 y 19 caracteres, incluidos
        throw Incorrecto(Razon::LONGITUD);

    if(!luhn(n)) //si numero invalido segun el algoritmo de luhn
        throw Incorrecto(Razon::NO_VALIDO);

    numero_=n;
}
Numero::~Numero(){}//destructor

///OPERADORES///
Numero::operator const char*()const {return numero_.c_str();}

///INCORRECTO///
Numero::Incorrecto::Incorrecto(const Numero::Razon r):r_(r){}
Numero::Razon Numero::Incorrecto::razon()const{return r_;}

///FUNCIONES///
void eliminarChar(Cadena& cad, size_t pos){ //fuera de clase
    Cadena nuevo = cad.substr(0, pos);

    if ((pos + 1) < cad.length())
        nuevo += Cadena(cad.substr(pos+1, cad.length()));

    cad = move(nuevo);
}

///////////////////////////////
////CLASE *TARJETA*///-----
//-----
///CONSTRUCTORES///
Tarjeta::Tarjeta( const Numero& n, Usuario& u, const Fecha& f):numero_(n), usuario_(&u), caducidad_(f), activa_(1), tipo_(esTipo()){
    if (f < Fecha())throw Caducada(f);
    usuario_->es_titular_de(*this);
}
```



WUOLAH

```

}

Tarjeta::~Tarjeta(){//destructor
    if(usuario_) usuario_->no_es_titular_de(*this);
}

////OPERADORES////
std::ostream& operator <<(std::ostream& os, const Tarjeta& t)
{
    os << t.tipo() << std::endl << t.numero() << std::endl;
    //Transformar nombre y apellidos en mayúsculas
    Cadena aux = t.titular()->nombre(); //Se asigna nombre a una cadena auxiliar modificable
    int i=0;
    while(aux[i]!='\0')
        {if (islower(aux[i])) aux[i]=toupper(aux[i]);i++;}
    os << aux << " ";

    i=0;
    aux = t.titular()->apellidos(); //Se asigna apellidos a una cadena auxiliar modificable
    while(aux[i]!='\0')
        {if (islower(aux[i])) aux[i]=toupper(aux[i]);i++;}
    os << aux << std::endl;

    //Se formatea la fecha para mostrarla
    os << "Caduca: " << std::setfill ('0') << std::setw (2) << t.caducidad().mes() << "/" << std::setw (2) << (t.caducidad().anno() % 100) << std::endl;
    return os;
}

ostream& operator <<(ostream& os, const Tarjeta::Tipo& tipo){
    switch(tipo){
        case Tarjeta::VISA: os << "VISA"; break;
        case Tarjeta::Mastercard: os << "Mastercard"; break;
        case Tarjeta::Maestro: os << "Maestro"; break;
        case Tarjeta::JCB: os << "JCB"; break;
        case Tarjeta::AmericanExpress: os << "AmericanExpress"; break;
        case Tarjeta::Otro: os << "Otro"; break;
        default: os << "Error";
    }
    return os;
}
bool operator <(const Numero& a, const Numero& b){return strcmp(a, b) < 0;}
bool operator <(const Tarjeta& a, const Tarjeta& b){return a.numero() < b.numero();}
bool operator >(const Tarjeta& a, const Tarjeta& b){return b.numero() < a.numero();}

////OBSERVADORES///

bool& Tarjeta::activa(bool a){activa_=a; return activa_;}

////CADUCADA////
Tarjeta::Caducada::Caducada(const Fecha& f):fecha_caducada(f){}

////NUM_DUPPLICADO////
Tarjeta::Num_duplicado::Num_duplicado(const Numero& n):numer_(n){}
const Numero& Tarjeta::Num_duplicado::que()const{return numer_;}

////DESACTIVADA////
Tarjeta::Desactivada::Desactivada(){}

////FUNCIONES////
void Tarjeta::anula_titular(){
    activa_=false;
    const_cast<Usuario*&>(usuario_) =nullptr;
}

```

```

Tarjeta::Tipo Tarjeta::esTipo(){
    int a = atoi(numero_.num().substr(0, 2).c_str());
    switch(a/10){
        case 3:
            if(a==34 || a==37) return Tarjeta::AmericanExpress;
            else return Tarjeta::JCB;
            break;
        case 4:
            return Tarjeta::VISA;
            break;
        case 5:
            return Tarjeta::Mastercard;
            break;
        case 6:
            return Tarjeta::Maestro;
            break;
        default:
            return Tarjeta::Otro;
    }
}

```

```

usuario.hpp
#ifndef USUARIO_HPP
#define USUARIO_HPP

#include "cadena.hpp"
#include "tarjeta.hpp"
#include "articulo.hpp"
#include <map>
#include <set>
#include <unordered_map>
#include <unordered_set>
#include <unistd.h>

class Numero;
class Tarjeta;
///////////////////////////////
///CLASE *CLAVE*///-----
//-----

class Clave{
    private:
        ///VARIABLES///
        Cadena pass_c;

    public:
        ///CONSTRUCTORES///
        Clave(const char*); 
        ~Clave(); //destructor

        ///OBSERVADORES///
        Cadena clave() const; //devuelve la contraseña cifrada

        ///CONSTANTES///
        enum Razon{CORTA, ERROR_CRYPT};

        ///FUNCIONES///
        bool verifica (const char*)const;

        ///INCORRECTA///
        class Incorrecta {
            public:
                Incorrecta(const Clave::Razon r);
                Clave::Razon razon() const;
            private:
                Clave::Razon r_;
        };
    protected:
};

///////////////////////////////
///CLASE *USUARIO*///-----
//-----


class Usuario{
    public:
        ///CONSTRUCTORES///
        Usuario(const Cadena& _id, const Cadena& _nomb, const Cadena& _apell, const Cadena& _direcc, const Clave& _pass);
        Usuario(const Usuario& that) = delete;
        Usuario& operator=(const Usuario &)=delete;
        ~Usuario();//destructor

        ///CONSTANTES///
        typedef std::map<Numero, Tarjeta*> Tarjetas;
        typedef std::unordered_map<Articulo*,unsigned> Articulos;
}

```

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



```
////ID_DUPPLICADO////
class Id_duplicado{
public:
    Id_duplicado(const Cadena&);
    const Cadena& idd()const; //observador
private:
    Cadena idd_;
};

////OBSERVADORES////
const Cadena& id() const noexcept{return ID_;}
const Cadena& nombre() const noexcept{return nombre_;}
const Cadena& apellidos() const noexcept{return apell_;}
const Cadena& direccion() const noexcept{return direcc_;}
const Tarjetas& tarjetas() const noexcept{return tarjetas_;}
const size_t n_articulos()const noexcept{return n_articulos_;}
const Articulos& compra() const noexcept{return articulos_;}

////FUNCIONES////
void es_titular_de(const Tarjeta&);           //se le asocia una tarjeta
void no_es_titular_de(Tarjeta&);                //se desasocia una tarjeta
void compra(const Articulo& a, size_t cant=1);

friend std::ostream& operator << (std::ostream& os, const Usuario &u) noexcept;

private:
    ////VARIABLES////
    Cadena ID_, nombre_, apell_, direcc_;
    Clave pass_;
    Tarjetas tarjetas_;
    Articulos articulos_;
    size_t n_articulos_;
    static std::unordered_set<Cadena> registrados;

protected:
};

std::ostream& operator << (std::ostream& os, const Usuario& u) noexcept;
void mostrar_carro (std::ostream& os,const Usuario& u);
#endif
```



```

usuario.cpp
#include "usuario.hpp" // CONTIENE CLASE *USUARIO* Y CLASE *CLAVE*
#include "tarjeta.hpp"
#include "articulo.hpp"

#include <unistd.h>
#include <cstdlib>
#include <string.h>
#include <random>
#include <iomanip>
#include <set>

using namespace std;
unordered_set<Cadena> Usuario::registrados;
///////////////////////////////
///CLASE *CLAVE*///-----
//-----
///CONSTRUCTOR///
Clave::Clave(const char* p){
    setlocale(LC_ALL, "");
    if(strlen(p)<5) throw Clave::Incorrecta(Clave::CORITA); //ERROR CORTA

    random_device r;
    uniform_int_distribution<size_t> dist(0, 63);
    char const MD5chars[] = "./ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
    char salt[2] = {MD5chars[dist(r)],MD5chars[dist(r)]};

    if(crypt(p, salt) == nullptr) throw Clave::Incorrecta(Clave::ERROR_CRYPT); //ERROR ERROR_CRYPT
    pass_c = crypt(p, salt);
}
Clave::~Clave(){           //destructor
}

///INCORRECTA///
Clave::Incorrecta::Incorrecta(const Clave::Razon r):r_(r){
}
Clave::Razon Clave::Incorrecta::razon() const {return r_;}

///FUNCIONES///
bool Clave::verifica(const char* c) const {
    char * p=crypt(c, pass_c.c_str());
    return p == pass_c;
}

///OBSERVADORES///
Cadena Clave::clave() const {return pass_c; }

/////////////////////////////
///CLASE *USUARIO*///-----
//-----
///CONSTRUCTORES///
Usuario::Usuario(const Cadena& _id, const Cadena& _nomb, const Cadena& _apell, const Cadena& _direcc, const Clave& _pass)
:_ID(_id), nombre(_nomb), apell(_apell), direcc(_direcc), pass(_pass), n_articulos_(articulos_.size()){
    if(!registrados.insert(ID_).second)
        throw Id_duplicado(ID_);
}
Usuario::~Usuario(){
    for(auto& i : tarjetas_)
        i.second->anula_titular();
    registrados.erase(ID_); //borra el id del usuario
}

///ID DUPLICADO///
Usuario::Id_duplicado::Id_duplicado(const Cadena& c):idd_(c){}
const Cadena& Usuario::Id_duplicado::idd() const {return idd_;} //observador

```

```

///FUNCIONES///
void Usuario::es_titular_de(const Tarjeta &t){
    if(t.titular() == this) tarjetas_[t.numero()] = const_cast<Tarjeta*>(&t);
}
void Usuario::no_es_titular_de(Tarjeta &t){tarjetas_.erase(t.numero());}

void Usuario::compra(const Articulo& a, size_t cant){
    Usuario::Articulos::const_iterator got = articulos_.find (const_cast<Articulo*>(&a));

    if (got == articulos_.end() ){
        if(cant > 0){
            articulos_[const_cast<Articulo*>(&a)] = cant;
            n_articulos_++;
        }
    }else{
        if(cant > 0 ){
            articulos_[const_cast<Articulo*>(&a)] = cant;
        }else{
            articulos_.erase(const_cast<Articulo*>(&a));
            n_articulos--;
        }
    }
}

```

///OPERADORES///

```

std::ostream& operator << (std::ostream& os, const Usuario& u) noexcept{
    setlocale(LC_ALL, "");
    os<<u.ID_<<"["<<u.pass_clave()<<"] "<<u.nombre_<<" "<<u.apell_<<endl;
    os<<u.direcc_<<endl;
    os<<"Tarjetas:\n";
    for (auto& t : u.tarjetas_)
        os << *t.second << endl;
    return os;
}
void mostrar_carro (ostream& os, const Usuario& u){

    os<<"Carrito de compra de "<<u.id()<<" [Artículos: "<<u.n_articulos()<<"]"<< endl<< " Cant.\tArtículo" <<endl;
    os <<"=====*><======"<<endl;

    for(Usuario::Articulos::const_iterator i = u.compra().begin(); i != u.compra().end(); i++){
        os << " " << i->second << "\t" << "[" << i->first->referencia() << "] \" " << i->first->titulo() << "\", ";
        os << i->first->f_publi().anno() << ". " << setprecision(2) << std::fixed << i->first->precio() << " €" << endl;
    }
}

```

P3

Mejora de tarjeta(numero) y 3 clases nuevas

```
articulo.hpp
#ifndef ARTICULO_HPP
#define ARTICULO_HPP

#include "fecha.hpp"
#include "cadena.hpp"

class Articulo{
    private:
        Cadena referencia_;
        Cadena titulo_;
        Fecha f_publi_;
        double precio_;
        unsigned stock_;
    public:
        //CONSTRUCTOR///
        Articulo(const Cadena& r, const Cadena& t, const Fecha& f, double p, unsigned s);

        //OBSERVADORES///
        Cadena referencia() const;
        Cadena titulo() const;
        Fecha f_publi() const;
        double precio() const;
        unsigned stock() const;
        double& precio();
        unsigned& stock();
    protected:
};

std::ostream& operator <<(std::ostream& os, const Articulo& ar)noexcept;

#endif
```

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



```
articulo.cpp
#include "articulo.hpp"
#include "fecha.hpp"
#include "cadena.hpp"
#include <iomanip>
#include <stdlib.h>
#include <iostream>
#include <locale>

using namespace std;

///CONSTRUCTOR///
Articulo::Articulo(const Cadena& r, const Cadena& t, const Fecha& f, double p, unsigned s)
:referencia_(r), titulo_(t), f_publi_(f), precio_(p), stock_(s){}

///OBSERVADORES///
Cadena Articulo::referencia()const {return referencia_;}
Cadena Articulo::titulo() const {return titulo_;}
Fecha Articulo::f_publi() const {return f_publi_;}
double Articulo::precio() const {return precio_;}
unsigned Articulo::stock() const {return stock_;}
double& Articulo::precio() {return precio_;}
unsigned&Articulo::stock() {return stock_;}

std::ostream& operator <<(std::ostream& os, const Articulo& ar)noexcept{
    std::locale::global(std::locale( "" ));
    os<<"["<<ar.referencia()<<"]\"";
    os<<ar.titulo()<<"\", "<<ar.f_publi().anno()<<". "<<fixed<<setprecision(2)<<ar.precio()<< " € ";
    return os;
}
```



```

tarjeta.hpp
#ifndef TARJETA_HPP
#define TARJETA_HPP
#include "cadena.hpp"
#include "tarjeta.hpp"
#include "articulo.hpp"
#include "usuario.hpp"

class Usuario;
class Clave;
///////////////////////////////
///CLASE *NUMERO*///-----
//-----
class Numero{
    private:
        ///VARIABLES///
        Cadena numero_;

    public:
        ///CONSTANTES///
        enum Razon {LONGITUD, DIGITOS, NO_VALIDO};

        ///CONSTRUCTORES///
        Numero(const Cadena&);
        ~Numero();//destructor

        ///OPERADORES///
        operator const char*()const;

        ///OBSERVADORES///
        const Cadena& num() const{return numero_}

        ///INCORRECTO///
        class Incorrecto{
            public:
                Incorrecto(const Numero::Razon r);
                Numero::Razon razon()const;
            private:
                Numero::Razon r_;
        };
};

void eliminarChar(Cadena& cad, size_t pos);

///////////////////////////////
///CLASE *TARJETA*///-----
//-
class Tarjeta{
    public:
        ///CONSTANTES///
        enum Tipo {Otro, VISA, Mastercard, Maestro, JCB, AmericanExpress};

        ///CONSTRUCTORES///
        Tarjeta(const Numero& n, Usuario& usuario, const Fecha& caducidad);
        Tarjeta(const Tarjeta& that) = delete;
        Tarjeta& operator=(const Tarjeta &)=delete;
        ~Tarjeta();

        ///OBSERVADORES///
        const Tarjeta::Tipo& tipo() const{return tipo_}
        const Numero& numero() const{return numero_}
        Usuario* titular() const{return usuario_}
        const Fecha& caducidad() const{return caducidad_}
        const bool activa()const{return activa_}

        bool &activa(bool a=true);
};

```

```

////CADUCADA////
class Caducada{
public:
    Caducada(const Fecha& f);
    const Fecha& cuando()const{return fecha_caducada;}
private:
    Fecha fecha_caducada;
};

////NUM_DUPPLICADO////
class Num_duplicado{
public:
    Num_duplicado(const Numero& );
    const Numero& que()const;
private:
    Numero numer_;
};

////DESACTIVADA////
class Desactivada{
public:
    Desactivada();
private:
};

////FUNCIONES////
void anula_titular();
Tipo esTipo();

friend std::ostream& operator<<(std::ostream& os, const Tarjeta& t);

private:
    ////VARIABLES////
    Numero numero_;           //numero troquelado
    Usuario* const usuario_;   //su titular
    Fecha caducidad_;         //caducidad
    bool activa_;              //1 == activa ; 0==no
    Tipo tipo_;                //tipo de la tarjeta
};

std::ostream& operator <<(std::ostream& os, const Tarjeta& t);
std::ostream& operator <<(std::ostream& os, const Tarjeta::Tipo& tipo);
bool operator <(const Numero& a, const Numero& b);
bool operator <(const Tarjeta& a, const Tarjeta& b);
bool operator >(const Tarjeta& a, const Tarjeta& b);
#endif

```

```

tarjeta.cpp
#include "tarjeta.hpp"
#include <iomanip>
#include <cstring>
#include <string.h>
#include <cctype>
#include <algorithm> // std::remove_if
using namespace std;
//algoritmo de luhn
bool luhn(const Cadena&);

///////////////////////////////
///CLASE *NUMERO*///-----
//-----
///CONSTRUCTORES///
class EsDigito: public unary_function<char,bool>
{
public:
    bool operator()(char c) const {return isdigit(c);}
};

Numero::Numero(const Cadena& num){
    Cadena n(num);
    Cadena::iterator i = std::remove_if (n.begin(), n.end(), [](char x){return std::isspace(x);});
    if(i != n.end()){
        *i = '\0'; //Si se ha borrado algún espacio en blanco
        n = Cadena(n.c_str()); //Marca el nuevo final de la cadena
        //Actualiza el tamaño de la cadena después de eliminar los espacios
    }

    unary_negate<EsDigito> not_EsDigito((EsDigito()));
    Cadena::const_iterator it = find_if(n.begin(), n.end(), not_EsDigito);
    if(it!=n.end()) throw Incorrecto(Razon::DIGITOS);

    if(n.length()>19 || n.length()<13) //si no esta entre 13 y 19 caracteres, incluidos
        throw Incorrecto(Razon::LONGITUD);

    if(!luhn(n)) //si numero invalido segun el algoritmo de luhn
        throw Incorrecto(Razon::NO_VALIDO);

    numero_=n;
}
Numero::~Numero(){}//destructor

///OPERADORES///
Numero::operator const char*() const {return numero_.c_str();}

///INCORRECTO///
Numero::Incorrecto::Incorrecto(const Numero::Razon r):r_(r){}
Numero::Razon Numero::Incorrecto::razon() const{return r_;}

///FUNCIONES///
void eliminarChar(Cadena& cad, size_t pos){ //fuera de clase
    Cadena nuevo = cad.substr(0, pos);

    if ((pos + 1) < cad.length())
        nuevo += Cadena(cad.substr(pos+1, cad.length()));

    cad = move(nuevo);
}

/////////////////////////////
///CLASE *TARJETA*///-----
//-----
///CONSTRUCTORES///
Tarjeta::Tarjeta( const Numero& n, Usuario& u, const Fecha& f):numero_(n), usuario_(&u), caducidad_(f), activa_(1), tipo_(esTipo()){
    if (f < Fecha())throw Caducada(f);
    usuario_->es_titular_de(*this);
}

```

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



```
}

Tarjeta::~Tarjeta(){//destructor
    if(usuario_) usuario_->no_es_titular_de(*this);
}

///OPERADORES///
std::ostream& operator <<(std::ostream& os, const Tarjeta& t)
{

    os << t.tipo() << std::endl << t.numero() << std::endl;
    //Transformar nombre y apellidos en mayúsculas
    Cadena aux = t.titular()->nombre(); //Se asigna nombre a una cadena auxiliar modificable
    int i=0;
    while(aux[i]!='\0')
        {if (islower(aux[i])) aux[i]=toupper(aux[i]);i++;}
    os << aux << " ";

    i=0;
    aux = t.titular()->apellidos(); //Se asigna apellidos a una cadena auxiliar modificable
    while(aux[i]!='\0')
        {if (islower(aux[i])) aux[i]=toupper(aux[i]);i++;}
    os << aux << std::endl;

    //Se formatea la fecha para mostrarla
    os << "Caduca: " << std::setfill ('0') << std::setw (2) << t.caducidad().mes() << "/" << std::setw (2) << (t.caducidad().anno() % 100) << std::endl;
    return os;
}

ostream& operator <<(ostream& os, const Tarjeta::Tipo& tipo){
switch(tipo){
    case Tarjeta::VISA: os << "VISA"; break;
    case Tarjeta::Mastercard: os << "Mastercard"; break;
    case Tarjeta::Maestro: os << "Maestro"; break;
    case Tarjeta::JCB: os << "JCB"; break;
    case Tarjeta::AmericanExpress: os << "AmericanExpress"; break;
    case Tarjeta::Otro: os << "Otro"; break;
    default: os << "Error";
}
return os;
}

bool operator <(const Numero& a, const Numero& b){return strcmp(a, b) < 0;}
bool operator <(const Tarjeta& a, const Tarjeta& b){return a.numero() < b.numero();}
bool operator >(const Tarjeta& a, const Tarjeta& b){return b.numero() < a.numero();}

///OBSERVADORES///

bool& Tarjeta::activa(bool a){activa_=a; return activa_;}

```

///CADUCADA///

```
Tarjeta::Caducada::Caducada(const Fecha& f):fecha_caducada(f){}
```

///NUM_DUPPLICADO///

```
Tarjeta::Num_duplicado::Num_duplicado(const Numero& n):numer_(n){}
const Numero& Tarjeta::Num_duplicado::que()const{return numer_;}
```

///DEACTIVADA///

```
Tarjeta::Desactivada::Desactivada(){}

```

///FUNCIONES///

```
void Tarjeta::anula_titular(){
    activa_=false;
    const_cast<Usuario*&>(usuario_) =nullptr;
}
```



WUOLAH

```

Tarjeta::Tipo Tarjeta::esTipo(){
    int a = atoi(numero_.num().substr(0, 2).c_str());
    switch(a/10){
        case 3:
            if(a==34 || a==37) return Tarjeta::AmericanExpress;
            else return Tarjeta::JCB;
            break;
        case 4:
            return Tarjeta::VISA;
            break;
        case 5:
            return Tarjeta::Mastercard;
            break;
        case 6:
            return Tarjeta::Maestro;
            break;
        default:
            return Tarjeta::Otro;
    }
}

```

```

usuario.hpp
#ifndef USUARIO_HPP
#define USUARIO_HPP

#include "cadena.hpp"
#include "tarjeta.hpp"
#include "articulo.hpp"
#include <map>
#include <set>
#include <unordered_map>
#include <unordered_set>
#include <unistd.h>

class Numero;
class Tarjeta;
///////////////////////////////
///CLASE *CLAVE*///-----
//-----

class Clave{
    private:
        ///VARIABLES///
        Cadena pass_c;

    public:
        ///CONSTRUCTORES///
        Clave(const char*); 
        ~Clave(); //destructor

        ///OBSERVADORES///
        Cadena clave() const; //devuelve la contraseña cifrada

        ///CONSTANTES///
        enum Razon{CORTA, ERROR_CRYPT};

        ///FUNCIONES///
        bool verifica (const char*)const;

        ///INCORRECTA///
        class Incorrecta {
            public:
                Incorrecta(const Clave::Razon r);
                Clave::Razon razon() const;
            private:
                Clave::Razon r_;
        };
    protected:
};

///////////////////////////////
///CLASE *USUARIO*///-----
//-----


class Usuario{
    public:
        ///CONSTRUCTORES///
        Usuario(const Cadena& _id, const Cadena& _nomb, const Cadena& _apell, const Cadena& _direcc, const Clave& _pass);
        Usuario(const Usuario& that) = delete;
        Usuario& operator=(const Usuario &)=delete;
        ~Usuario();//destructor

        ///CONSTANTES///
        typedef std::map<Numero, Tarjeta*> Tarjetas;
        typedef std::unordered_map<Articulo*,unsigned> Articulos;
}

```

```

////ID_DUPPLICADO////
class Id_duplicado{
    public:
        Id_duplicado(const Cadena&);
        const Cadena& idd()const; //observador
    private:
        Cadena idd_;
};

////OBSERVADORES////
const Cadena& id() const noexcept{return ID_;}
const Cadena& nombre() const noexcept{return nombre_;}
const Cadena& apellidos() const noexcept{return apell_;}
const Cadena& direccion() const noexcept{return direcc_;}
const Tarjetas& tarjetas() const noexcept{return tarjetas_;}
const size_t n_articulos()const noexcept{return n_articulos_;}
const Articulos& compra() const noexcept{return articulos_;}

////FUNCIONES////
void es_titular_de(const Tarjeta&);           //se le asocia una tarjeta
void no_es_titular_de(Tarjeta&);               //se desasocia una tarjeta
void compra(const Articulo& a, size_t cant=1);

friend std::ostream& operator << (std::ostream& os, const Usuario &u) noexcept;

private:
    ////VARIABLES///
    Cadena ID_, nombre_, apell_, direcc_;
    Clave pass_;
    Tarjetas tarjetas_;
    Articulos articulos_;
    size_t n_articulos_;
    static std::unordered_set<Cadena> registrados;

protected:
};

std::ostream& operator << (std::ostream& os, const Usuario& u) noexcept;
void mostrar_carro (std::ostream& os,const Usuario& u);
#endif

```

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



```
usuario.cpp
#include "usuario.hpp" // CONTIENE CLASE *USUARIO* Y CLASE *CLAVE*
#include "tarjeta.hpp"
#include "articulo.hpp"

#include <unistd.h>
#include <cstdlib>
#include <string.h>
#include <random>
#include <iomanip>
#include <set>

using namespace std;
unordered_set<Cadena> Usuario::registrados;
/////////////////////////////////////////////////////////////////
///CLASE *CLAVE*///
//-----
///CONSTRUCTOR///
Clave::Clave(const char* p){
    setlocale(LC_ALL, "");
    if(strlen(p)<5) throw Clave::Incorrecta(Clave::CORTA); //ERROR CORTA
    random_device r;
    uniform_int_distribution<size_t> dist(0, 63);
    char const MD5chars[] = "./ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
    char salt[2] = {MD5chars[dist(r)],MD5chars[dist(r)]};

    if(crypt(p, salt) == nullptr) throw Clave::Incorrecta(Clave::ERROR_CRYPT); //ERROR ERROR_CRYPT
    pass_c = crypt(p, salt);
}
Clave::~Clave(){ //destructor
}

///INCORRECTA///
Clave::Incorrecta::Incorrecta(const Clave::Razon r):r_(r){
}
Clave::Razon Clave::Incorrecta::razon() const {return r_;}

///FUNCIONES///
bool Clave::verifica(const char* c) const {
    char * p=crypt(c, pass_c.c_str());
    return p == pass_c;
}

///OBSERVADORES///
Cadena Clave::clave() const {return pass_c;}

/////////////////////////////////////////////////////////////////
///CLASE *USUARIO*///
//-----
///CONSTRUCTORES///
Usuario::Usuario(const Cadena& _id, const Cadena& _nomb, const Cadena& _apell, const Cadena& _direcc, const Clave& _pass)
:_ID(_id), nombre(_nomb), apell(_apell), direcc(_direcc), pass(_pass), n_articulos_(articulos_.size()){
    if(!registrados.insert({_ID}.second))
        throw Id_duplicado({_ID});
}
Usuario::~Usuario(){
    for(auto& i : tarjetas_)
        i.second->anula_titular();
    registrados.erase({_ID}); //borra el id del usuario
}

///ID_DUPPLICADO///
Usuario::Id_duplicado::Id_duplicado(const Cadena& c):idd_(c){}
const Cadena& Usuario::Id_duplicado::idd() const {return idd_;} //observador
```



WUOLAH

```

///FUNCIONES///
void Usuario::es_titular_de(const Tarjeta &t){
    if(t.titular() == this) tarjetas_[t.numero()] = const_cast<Tarjeta*>(&t);
}
void Usuario::no_es_titular_de(Tarjeta &t){tarjetas_.erase(t.numero());}

void Usuario::compra(const Articulo& a, size_t cant){
    Usuario::Articulos::const_iterator got = articulos_.find (const_cast<Articulo*>(&a));

    if (got == articulos_.end() ){
        if(cant > 0){
            articulos_[const_cast<Articulo*>(&a)] = cant;
            n_articulos_++;
        }
    }else{
        if(cant > 0 ){
            articulos_[const_cast<Articulo*>(&a)] = cant;
        }else{
            articulos_.erase(const_cast<Articulo*>(&a));
            n_articulos--;
        }
    }
}

```

///OPERADORES///

```

std::ostream& operator << (std::ostream& os, const Usuario& u) noexcept{
    setlocale(LC_ALL, "");
    os<<u.ID_<<"["<<u.pass_clave()<<"] "<<u.nombre_<<" "<<u.apell_<<endl;
    os<<u.direcc_<<endl;
    os<<"Tarjetas:\n";
    for (auto& t : u.tarjetas_)
        os << *t.second << endl;
    return os;
}
void mostrar_carro (ostream& os, const Usuario& u){

    os<<"Carrito de compra de "<<u.id()<<" [Artículos: "<<u.n_articulos()<<"]"<< endl<< " Cant.\tArtículo" <<endl;
    os <<"======"<< endl;

    for(Usuario::Articulos::const_iterator i = u.compra().begin(); i != u.compra().end(); i++){
        os << " " << i->second << "\t" << "[" << i->first->referencia() << "] \" " << i->first->titulo() << "\", ";
        os << i->first->f_publi().anno() << ". " << setprecision(2) << std::fixed << i->first->precio() << " €" << endl;
    }
}

```

```

pedido.hpp
#ifndef PEDIDO_HPP_
#define PEDIDO_HPP_

#include "usuario.hpp"
#include "tarjeta.hpp"
#include "fecha.hpp"
#include "articulo.hpp"
#include "pedido-articulo.hpp"
#include "usuario-pedido.hpp"
#include<iomanip>
#include<iostream>

class Pedido_Articulo;
class Tarjeta;
class Usuario_Pedido;
class Pedido {
public:
    //CONSTRUCTOR///
    Pedido(Usuario_Pedido& UP, Pedido_Articulo& PA, Usuario& usuario, const Tarjeta& tarjeta, const Fecha& f= Fecha());
    //OBSERVADORES///
    const int numero() const noexcept {return n_p_;}
    const Tarjeta* tarjeta() const noexcept {return tarjeta_;}
    const Fecha fecha() const noexcept {return fPedido_;}
    const double total() const noexcept {return importe_;}
    static unsigned n_total_pedidos() noexcept {return N_pedidos;}

    //NACIO///
    class Vacio{
        Usuario* us_;
    public:
        Vacio(Usuario* us): us_(us){}
        Usuario& usuario() const{return *us_;}
    };

    //IMPOSTOR///
    class Impostor{
        Usuario* us_;
    public:
        Impostor(Usuario* us): us_(us){}
        Usuario& usuario() const{return *us_;}
    };

    //SIN STOCK///
    class SinStock{
        Articulo* art_;
    public:
        SinStock(Articulo* art): art_(art){}
        Articulo& articulo() const{return *art_;}
    };

private:
    //VARIABLES///
    int n_p_; //numero de pedido, desde 1, incrementandose en 1
    const Tarjeta* tarjeta_; //tarjeta con la que se paga
    Fecha fPedido_; //fecha del pedido
    double importe_; //importe total
    static unsigned N_pedidos; //cantidad de pedidos hechos
protected:
};

std::ostream& operator<<(std::ostream& os, const Pedido& p);
#endif

```

```

pedido.cpp
#include "pedido.hpp"
using namespace std;

unsigned Pedido::N_pedidos = 0;

///CONSTRUCTOR///
Pedido::Pedido(Usuario_Pedido& UP, Pedido_Articulo& PA, Usuario& usuario,const Tarjeta& tarjeta, const Fecha& f)
:n_p_(N_pedidos+1),tarjeta_(&tarjeta),fPedido_(f),importe_(0.) {
    if(usuario.compra().empty())
        throw Pedido::Vacio(&usuario);
    if(tarjeta.titular() != &usuario)
        throw Pedido::Impostor(&usuario);
    if(tarjeta.caducidad() < f)
        throw Tarjeta::Caducada(tarjeta.caducidad());
    if( !tarjeta.activa() )
        throw Tarjeta::Desactivada();

    auto carro = usuario.compra();
    for(auto& iter : carro) {
        if ( iter.first->stock() < iter.second ){
            for(auto&i : carro)
                usuario.compra(*i.first, 0);
            //usuario.compra().clear();
            throw Pedido::SinStock(iter.first);
        }
    }
    UP.asocia(*this, usuario);
    for(auto& iter : carro) {
        importe_ += iter.first->precio() * iter.second;
        PA.pedir(*iter.first, *this, iter.first->precio(), iter.second);
        iter.first->stock()-=iter.second;
    }
}

const_cast<Usuario::Articulos&>(usuario.compra()).clear();
n_p_ = ++ N_pedidos;
}

```

```

///OPERADOR DE INSERCCION///
std::ostream& operator<<(std::ostream& os, const Pedido& p){

    os << left;
    os << "Núm. pedido: " << p.numero() << endl;
    os << setw(13) << "Fecha: " << p.fecha() << endl;
    os << setw(13) << "Pagado con: " << p.tarjeta()->tipo() << " n.º: " << p.tarjeta()->numero() << endl;
    os << setw(13) << "Importe: " << fixed << setprecision(2) << p.total() << " €" << endl;

    return os;
}

```

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



```
pedido-articulo.hpp
#ifndef Pedido_Articulo_HPP_
#define Pedido_Articulo_HPP_

#include<iostream>
#include <iomanip>
#include "articulo.hpp"
#include "pedido.hpp"

///////////////CLASE *LineaPedido*////-----
//-----
class LineaPedido{
public:
    ///CONSTRUCTOR///
    explicit LineaPedido(double pr, unsigned c=1);
    ///OBSERVADORES///
    const double precio_venta() const noexcept {return precio_;}
    const unsigned cantidad() const noexcept {return cantidad_;}
private:
    ///VARIABLES///
    double precio_;
    unsigned cantidad_;
};

std::ostream& operator<<(std::ostream& os, const LineaPedido& p);

class Pedido;
class Articulo;
///FUNCIONES DE CLASE///
class OrdenaPedidos: public std::binary_function<Pedido*,Pedido*,bool>{
public:
    bool operator()(const Pedido* p1,const Pedido* p2) const;
};

class OrdenaArticulos: public std::binary_function<Articulo*,Articulo*,bool>{
public:
    bool operator()(const Articulo* a1,const Articulo* a2) const {return (a1->referencia() < a2->referencia());}
};

////////////CLASE *Pedido_Articulo*////-----
//-----
class Pedido_Articulo{
public:
    ///LIBRERIAS///
    typedef std::map<Articulo*, LineaPedido, OrdenaArticulos> ItemsPedido;
    typedef std::map<Pedido*, LineaPedido, OrdenaPedidos> Pedidos;
    typedef std::map<Pedido*, ItemsPedido, OrdenaPedidos> PedArt;
    typedef std::map<Articulo*, Pedidos, OrdenaArticulos> ArtPed;

    ///OBSERVADORES///
    //PEDIR/
    void pedir(Pedido& ped, Articulo& art, double precio, unsigned cantidad=1); //PEDIDO - ARTICULO
    void pedir(Articulo& art, Pedido& ped, double precio, unsigned cantidad=1); //ARTICULO - PEDIDO
    //DETALLE/
    const ItemsPedido& detalle(Pedido& p);
    //VENTAS/
    Pedidos ventas(Articulo& a);
    //mostrarDetallePedidos/
    std::ostream& mostrarDetallePedidos(std::ostream& os) const;
    //mostrarVentasArticulos/
    std::ostream& mostrarVentasArticulos(std::ostream& os) const;

private:
    ///VARIABLES///
    PedArt ped_art_;
    ArtPed art_ped_;
};


```



WUOLAH

```
std::ostream& operator <<(std::ostream& os,const Pedido_Articulo::ItemsPedido& ip);  
std::ostream& operator <<(std::ostream& os,const Pedido_Articulo::Pedidos& p);
```

```
#endif
```

```

pedido-articulo.cpp
#include "pedido-articulo.hpp"
using namespace std;

///////////////////////////////
////CLASE *LineaPedido*///-----
//-----
///CONSTRUCTOR///
LineaPedido::LineaPedido(double pr, unsigned c)
: precio_(pr), cantidad_(c){}

///OPERATOR <<///
std::ostream& operator<<(std::ostream& os, const LineaPedido& p){

    os<<fixed<<setprecision(2)<<p.precio_venta()<<" €\t"<<p.cantidad();
    return os;
}

///////////////////////////////
////CLASE *Pedido_Articulo*///-----
//-----
bool OrdenaPedidos::operator()(const Pedido* p1,const Pedido* p2) const{return (p1->numero() < p2->numero());}

//PEDIR//
void Pedido_Articulo::pedir (Pedido& ped, Articulo& art, double precio, unsigned cantidad ){
    ped_art_[&ped].insert(make_pair(&art,LineaPedido(precio,cantidad)));
    art_ped_[&art].insert(make_pair(&ped,LineaPedido(precio,cantidad)));
}
void Pedido_Articulo::pedir (Articulo&a, Pedido&p, double precio, unsigned cantidad ){
    pedir(p,a,precio,cantidad);
}
//DETALLE//
const Pedido_Articulo::ItemsPedido& Pedido_Articulo::detalle(Pedido&p){
    return ped_art_.find(&p)->second;
}
//VENTAS//
Pedido_Articulo::Pedidos Pedido_Articulo::ventas(Articulo&a){
    if(art_ped_.find(&a)!=art_ped_.end()){
        return art_ped_.find(&a)->second;
    }else{
        Pedido_Articulo::Pedidos vacio;
        return vacio;
    }
}
//mostrarDetallePedidos//
ostream& Pedido_Articulo::mostrarDetallePedidos(ostream&os) const{
    double precio = 0.0;

    for(auto& iter : ped_art_){
        os << "Pedido núm. " << iter.first->numero();
        os << "\tCliente: " << iter.first->tarjeta()->titular()->nombre() << "\n";
        os << "Fecha: " << iter.first->fecha() << iter.second;
        precio += iter.first->total();
    }

    os << "TOTAL VENTAS: " << fixed<<setprecision(2) << precio << " €" << endl;
    return os;
}
//mostrarVentasArticulos//
ostream& Pedido_Articulo::mostrarVentasArticulos(ostream&os) const{
    for(auto& iter: art_ped_){
        os << "Ventas de " << "[" << iter.first->referencia() << "]";
        os << "\"" << iter.first->titulo() << "\" \n" << iter.second << endl;
    }
    return os;
}

```

```

///OPERADORES <>///
ostream& operator <<(ostream&os,const Pedido_Articulo::ItemsPedido& ip)
{
    double precio = 0;
    Pedido_Articulo::ItemsPedido::const_iterator i;
    os << endl << "=====\\n" << endl;
    os << " PVP \t Cant. \t Articulo \\n";
    os << "=====\\n" << endl;
    for(i = ip.begin(); i != ip.end(); ++i){
        os << " " << i->second.precio_venta() << "\t";
        os << i->second.cantidad() << "\t";
        os << "[" << i->first->referencia() << "] ";
        os << "\"" << i->first->título() << "\"" << endl;
        precio = precio + i->second.precio_venta() * i->second.cantidad();
    }
    os << "=====\\n" << endl;
    os << fixed;
    os << setprecision(2) << precio << " €" << endl;
    return os;
}
ostream& operator <<(ostream&os,const Pedido_Articulo::Pedidos& pedidos)
{
    double precio = 0;
    unsigned total = 0;
    Pedido_Articulo::Pedidos::const_iterator i;
    os << "=====\\n" << endl;
    os << " PVP \t Cant. \t Fecha venta \\n";
    os << "=====\\n" << endl;
    for(auto iter : pedidos){
        os << " " << iter.second.precio_venta() << "\t";
        os << iter.second.cantidad() << "\t";
        os << iter.first->fecha() << endl;
        precio += (iter.second.precio_venta() * iter.second.cantidad());
        total += iter.second.cantidad();
    }
    os << "=====\\n" << endl;
    os << fixed;
    os << setprecision(2) << precio << " €\t" << total << endl ;
    return os;
}

```

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



```
usuario-pedido.hpp
#ifndef USUARIO_PEDIDO_HPP_
#define USUARIO_PEDIDO_HPP_

#include <map>
#include <set>

class Pedido;
class Usuario;

class Usuario_Pedido{
public:
    //DEFINICIONES///
    typedef std::set<Pedido*> Pedidos;
    typedef std::map<Usuario*, Pedidos> UsuarioP;
    typedef std::map<Pedido*, Usuario*> PedidoU;

    //OBSERVADORES///
    //ASOCIA//
    void asocia(Usuario& us, Pedido& ped){usu_ped_[&us].insert(&ped); ped_usu_[&ped] = &us;}
    void asocia(Pedido& ped, Usuario& us){asocia(us, ped);}
    //PEDIDOS//
    Pedidos& pedidos(Usuario& us){return usu_ped_.find(&us)->second;}
    //CLIENTE//
    Usuario* cliente(Pedido& ped){return ped_usu_.find(&ped)->second;}

private:
    //VARIABLES///
    UsuarioP usu_ped_;
    PedidoU ped_usu_;
protected:
};

#endif

/* El .cpp no hace falta, pero en la P4 sí será necesaria, aunque vacío*/
```



P4

Mejora de artículo añadiendo subclases (modifica pedido) y añade autor.

```
articulo.hpp
#ifndef ARTICULO_HPP
#define ARTICULO_HPP

#include "fecha.hpp"
#include "cadena.hpp"
#include <iomanip>
#include <set>
#include <stdlib.h>
#include <iostream>
#include <locale>

class Autor{
public:
    ///CONSTRUCTOR///
    Autor(const Cadena& nomb, const Cadena& apell, const Cadena& direcc);

    ///OBSERVADORES///
    const Cadena& nombre() const noexcept {return nombre_;}
    const Cadena& apellidos() const noexcept {return apellidos_;}
    const Cadena& direccion() const noexcept {return direccion_;}

private:
    Cadena nombre_;
    Cadena apellidos_;
    Cadena direccion_;

protected:
};

class Articulo{
public:
    ///DICCIONARIO AUTORES///
    typedef std::set <Autor*> Autores;

    ///CONSTRUCTOR///
    Articulo(const Autores& autores, const Cadena& referencia, const Cadena& titulo, const Fecha& fecha_publi, const double precio);

    ///OBSERVADORES///
    const Cadena referencia() const noexcept{ return referencia_; }
    const Cadena& titulo() const noexcept{ return titulo_; }
    const Fecha& f_publi() const noexcept{ return f_publi_; }
    const double precio() const noexcept{ return precio_; }
    double& precio() noexcept{ return precio_; }
    const Autores& autores() const noexcept{ return autores_; }

    ///AUTORES_VACIOS///
    class Autores_vacios{};

    ///OSTREAM///
    virtual void impresion_especifica(std::ostream&) const = 0;

    ///DESTRUCTOR///
    virtual ~Articulo(){} //destructor

private:
    const Autores autores_;
    Cadena referencia_;
    Cadena titulo_;
    Fecha f_publi_;
    double precio_;

protected:
};


```

```

std::ostream& operator <<(std::ostream& os, const Articulo& ar)noexcept;
class LibroDigital : public Articulo {

    public:
        ///CONSTRUCTOR///
        LibroDigital(const Autores& autores, const Cadena& referencia, const Cadena& titulo, const Fecha& fecha_publi, const
double precio, const Fecha& fExp);

        ///OBSERVADORES///
        const Fecha f_expir() const noexcept{ return f_expir_;}
        void impresion_especifica(std::ostream& os) const;

    private:
        const Fecha f_expir_;

    protected:
};

class ArticuloAlmacenable : public Articulo {

    public:
        ///CONSTRUCTOR///
        ArticuloAlmacenable(const Autores& autores, const Cadena& referencia, const Cadena& titulo, const Fecha& fecha_publi,
const double precio, unsigned stock=0);

        ///OBSERVADORES///
        const unsigned stock() const noexcept{ return stock_;}
        unsigned& stock() noexcept{ return stock_;}

    protected:
        unsigned stock_;

};

class Libro : public ArticuloAlmacenable{
    public:
        ///CONSTRUCTOR///
        Libro(const Autores& autores, const Cadena& referencia, const Cadena& titulo, const Fecha& fecha_publi, const double
precio, unsigned paginas, unsigned stock=0);

        ///OBSERVADORES///
        const unsigned n_pag() const noexcept {return n_pag_;}
        void impresion_especifica(std::ostream& os) const;

    private:
        const unsigned n_pag_; //numero de páginas

    protected:
};

class Cederron : public ArticuloAlmacenable{
    public:
        ///CONSTRUCTOR///
        Cederron(const Autores& autores, const Cadena& referencia, const Cadena& titulo, const Fecha& fecha_publi, const
double precio, unsigned tamano, unsigned stock=0);

        ///OBSERVADORES///
        const unsigned tam() const noexcept {return tam_;}
        void impresion_especifica(std::ostream& os) const;

    private:
        const unsigned tam_;

    protected:
};

#endif

```

```

articulo.cpp
#include "articulo.hpp"
#include "fecha.hpp"
#include "cadena.hpp"

using namespace std;
///////////////////////////////
///CLASE *Articulo*///-----
//-----
///CONSTRUCTOR///
Articulo::Articulo(const Autores& autores, const Cadena& referencia, const Cadena& titulo, const Fecha& fecha_publi, const double precio)
: autores_(autores), referencia_(referencia), titulo_(titulo), f_publi_(fecha_publi), precio_(precio){
    if(autores.empty()) throw Autores_vacios();
}

///OSTREAM///
std::ostream& operator <<(std::ostream& os, const Articulo& ar)noexcept{
    std::locale::global( std::locale( "" ) );

    os<<"["<<ar.referencia()<<"] \"";
    os<<ar.titulo()<<"\", de ";
    const auto& autores = ar.autores();
    auto i = autores.begin();
    os << (*i)->apellidos();
    while (++i != autores.end()){
        os << ", " << (*i)->apellidos();
    }

    os << ". " << ar.f_publi().anno() << ". " <<fixed<<setprecision(2)<<ar.precio()<< " €\n\t";
    ar.impresion_especifica(os);
    return os;
}

/////////////////////////////
///CLASE *LibroDigital*///-----
//-----
///CONSTRUCTOR///
LibroDigital::LibroDigital(const Autores& autores, const Cadena& referencia, const Cadena& titulo, const Fecha& fecha_publi,
const double precio, const Fecha& fExp)
: Articulo(autores, referencia, titulo, fecha_publi, precio), f_expir_(fExp){}

///OSTREAM ESP///
void LibroDigital::impresion_especifica(ostream& os) const{
    os << "A la venta hasta el "<< f_expir_ << ".";
}

/////////////////////////////
///CLASE *ArticuloAlmacenable*///-----
//-----
///CONSTRUCTOR///
ArticuloAlmacenable::ArticuloAlmacenable(const Autores& autores, const Cadena& referencia, const Cadena& titulo, const Fecha&
fecha_publi, const double precio, unsigned stock)
: Articulo(autores, referencia, titulo, fecha_publi, precio), stock_(stock){}

/////////////////////////////
///CLASE *Libro*///-----
//-----
///CONSTRUCTOR///
Libro::Libro(const Autores& autores, const Cadena& referencia, const Cadena& titulo, const Fecha& fecha_publi, const double precio,
unsigned paginas, unsigned stock)
: ArticuloAlmacenable(autores, referencia, titulo, fecha_publi, precio, stock), n_pag_(paginas){}

///OSTREAM ESP///
void Libro::impresion_especifica(ostream& os) const{
    os << n_pag_ << " págs., " << stock_ << " unidades.";
}

```

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



```
///////////////  
///CLASE *Cederron*///-----  
//-----  
///CONSTRUCTOR///  
Cederron::Cederron(const Autores& autores, const Cadena& referencia, const Cadena& titulo, const Fecha& fecha_publi, const double precio,  
unsigned tamano, unsigned stock)  
: ArticuloAlmacenable(autores, referencia, titulo, fecha_publi, precio, stock), tam_(tamano){}  
  
///OSTREAM ESP///  
void Cederron::impresion_especifica(ostream& os) const{  
    os << tam_ << " MB, " << stock_ << " unidades.";  
}  
  
///////////////  
///CLASE *Autor*///-----  
//-----  
///CONSTRUCTOR///  
Autor::Autor(const Cadena& nomb, const Cadena& apell, const Cadena& direcc)  
: nombre_(nomb), apellidos_(apell), direccion_(direcc){}
```



```

pedido.hpp
#ifndef PEDIDO_HPP_
#define PEDIDO_HPP_

#include "usuario.hpp"
#include "tarjeta.hpp"
#include "fecha.hpp"
#include "articulo.hpp"
#include "pedido-articulo.hpp"
#include "usuario-pedido.hpp"
#include<iomanip>
#include<iostream>

class Pedido_Articulo;
class Tarjeta;
class Usuario_Pedido;
class Pedido {
public:
    //CONSTRUCTOR///
    Pedido(Usuario_Pedido& UP, Pedido_Articulo& PA, Usuario& usuario, const Tarjeta& tarjeta, const Fecha& f= Fecha());
    //OBSERVADORES///
    const int numero() const noexcept {return n_p_;}
    const Tarjeta* tarjeta() const noexcept {return tarjeta_;}
    const Fecha fecha() const noexcept {return fPedido_;}
    const double total() const noexcept {return importe_;}
    static unsigned n_total_pedidos() noexcept {return N_pedidos;}

    //NACIO///
    class Vacio{
        Usuario* us_;
    public:
        Vacio(Usuario* us): us_(us){}
        Usuario& usuario() const{return *us_;}
    };

    //IMPOSTOR///
    class Impostor{
        Usuario* us_;
    public:
        Impostor(Usuario* us): us_(us){}
        Usuario& usuario() const{return *us_;}
    };

    //SIN STOCK///
    class SinStock{
        Articulo* art_;
    public:
        SinStock(Articulo* art): art_(art){}
        Articulo& articulo() const{return *art_;}
    };

private:
    //VARIABLES///
    int n_p_; //numero de pedido, desde 1, incrementandose en 1
    const Tarjeta* tarjeta_; //tarjeta con la que se paga
    Fecha fPedido_; //fecha del pedido
    double importe_; //importe total
    static unsigned N_pedidos; //cantidad de pedidos hechos

protected:
};

std::ostream& operator<<(std::ostream& os, const Pedido& p);
#endif

```

```

pedido.cpp
#include "pedido.hpp"
using namespace std;

unsigned Pedido::N_pedidos = 0;

////CONSTRUCTOR////
Pedido::Pedido(Usuario_Pedido& UP, Pedido_Articulo& PA, Usuario& usuario,const Tarjeta& tarjeta, const Fecha& f)
:n_p_(N_pedidos+1),tarjeta_(&tarjeta),fPedido_(f),importe_(0.) {
    if(usuario.compra().empty())
        throw Pedido::Vacio(&usuario);
    if(tarjeta.titular() != &usuario)
        throw Pedido::Impostor(&usuario);
    if(tarjeta.caducidad() < f)
        throw Tarjeta::Caducada(tarjeta.caducidad());
    if( !tarjeta.activa() )
        throw Tarjeta::Desactivada();

    auto carro = usuario.compra();

    for(auto& iter : carro){
        if(ArticuloAlmacenable* AA = dynamic_cast<ArticuloAlmacenable*>(iter.first)) {
            if(iter.second > AA->stock()) {
                const_cast<Usuario::Articulos&>(usuario.compra()).clear();
                throw SinStock(iter.first);
            }
        } else
            if(LibroDigital* LD = dynamic_cast<LibroDigital*>(iter.first))
                if(LD->f_expir() < f)
                    usuario.compra(*iter.first, 0);
    }

    if(usuario.compra().empty())
        throw Vacio(&usuario); //Comprobamos si se ha quedado vacío

    UP.asocia(*this, usuario);
    carro = usuario.compra();

    for(auto& iter : carro) {
        importe_ += (iter.first->precio()) * (iter.second);
        PA.pedir(*this, *iter.first, iter.first->precio(), iter.second);
        if(ArticuloAlmacenable* AA = dynamic_cast<ArticuloAlmacenable*>(iter.first))
            AA->stock() -= iter.second;
    }

    const_cast<Usuario::Articulos&>(usuario.compra()).clear();
    n_p_ = ++ N_pedidos;
}

////OPERADOR DE INSERCCION////
std::ostream& operator<<(std::ostream& os, const Pedido& p){

    os << left;
    os << "Nºm. pedido: " << p.numero() << endl;
    os << setw(13) << "Fecha: " << p.fecha() << endl;
    os << setw(13) << "Pagado con: " << p.tarjeta()->tipo() << " nº: " << p.tarjeta()->numero() << endl;
    os << setw(13) << "Importe: " << fixed << setprecision(2) << p.total() << " €" << endl;

    return os;
}

```

```

tarjeta.hpp
#ifndef TARJETA_HPP
#define TARJETA_HPP
#include "cadena.hpp"
#include "tarjeta.hpp"
#include "articulo.hpp"
#include "usuario.hpp"

class Usuario;
class Clave;
///////////////////////////////
///CLASE *NUMERO*///-----
//-----
class Numero{
    private:
        ///VARIABLES///
        Cadena numero_;

    public:
        ///CONSTANTES///
        enum Razon {LONGITUD, DIGITOS, NO_VALIDO};

        ///CONSTRUCTORES///
        Numero(const Cadena&);
        ~Numero();//destructor

        ///OPERADORES///
        operator const char*()const;

        ///OBSERVADORES///
        const Cadena& num() const{return numero_}

        ///INCORRECTO///
        class Incorrecto{
            public:
                Incorrecto(const Numero::Razon r);
                Numero::Razon razon()const;
            private:
                Numero::Razon r_;
        };
};

void eliminarChar(Cadena& cad, size_t pos);

///////////////////////////////
///CLASE *TARJETA*///-----
//-
class Tarjeta{
    public:
        ///CONSTANTES///
        enum Tipo {Otro, VISA, Mastercard, Maestro, JCB, AmericanExpress};

        ///CONSTRUCTORES///
        Tarjeta(const Numero& n, Usuario& usuario, const Fecha& caducidad);
        Tarjeta(const Tarjeta& that) = delete;
        Tarjeta& operator=(const Tarjeta &)=delete;
        ~Tarjeta();

        ///OBSERVADORES///
        const Tarjeta::Tipo& tipo() const{return tipo_}
        const Numero& numero() const{return numero_}
        Usuario* titular() const{return usuario_}
        const Fecha& caducidad() const{return caducidad_}
        const bool activa()const{return activa_}

        bool &activa(bool a=true);
};

```

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



```
////CADUCADA////
class Caducada{
public:
    Caducada(const Fecha& f);
    const Fecha& cuando()const{return fecha_caducada;}
private:
    Fecha fecha_caducada;
};

////NUM_DUPPLICADO////
class Num_duplicado{
public:
    Num_duplicado(const Numero& );
    const Numero& que()const;
private:
    Numero numer_;
};

////DESACTIVADA////
class Desactivada{
public:
    Desactivada();
private:
};

////FUNCIONES////
void anula_titular();
Tipo esTipo();

friend std::ostream& operator<<(std::ostream& os, const Tarjeta& t);

private:
    ////VARIABLES////
    Numero numero_;          //numero troquelado
    Usuario* const usuario_;  //su titular
    Fecha caducidad_;        //caducidad
    bool activa_;              //1 == activa ; 0==no
    Tipo tipo_;                //tipo de la tarjeta
};

std::ostream& operator <<(std::ostream& os, const Tarjeta& t);
std::ostream& operator <<(std::ostream& os, const Tarjeta::Tipo& tipo);
bool operator <(const Numero& a, const Numero& b);
bool operator <(const Tarjeta& a, const Tarjeta& b);
bool operator >(const Tarjeta& a, const Tarjeta& b);
#endif
```



WUOLAH

```

tarjeta.cpp
#include "tarjeta.hpp"
#include <iomanip>
#include <cstring>
#include <string.h>
#include <cctype>
#include <algorithm> // std::remove_if
using namespace std;
//algoritmo de luhn
bool luhn(const Cadena&);

///////////////////////////////
///CLASE *NUMERO*///-----
//-----
///CONSTRUCTORES///
class EsDigito: public unary_function<char,bool>
{
public:
    bool operator()(char c) const {return isdigit(c);}
};

Numero::Numero(const Cadena& num){
    Cadena n(num);
    Cadena::iterator i = std::remove_if (n.begin(), n.end(), [](char x){return std::isspace(x);});
    if(i != n.end()){
        *i = '\0'; //Si se ha borrado algún espacio en blanco
        n = Cadena(n.c_str()); //Marca el nuevo final de la cadena
        //Actualiza el tamaño de la cadena después de eliminar los espacios
    }

    unary_negate<EsDigito> not_EsDigito((EsDigito()));
    Cadena::const_iterator it = find_if(n.begin(), n.end(), not_EsDigito);
    if(it!=n.end()) throw Incorrecto(Razon::DIGITOS);

    if(n.length()>19 || n.length()<13) //si no esta entre 13 y 19 caracteres, incluidos
        throw Incorrecto(Razon::LONGITUD);

    if(!luhn(n)) //si numero invalido segun el algoritmo de luhn
        throw Incorrecto(Razon::NO_VALIDO);

    numero_=n;
}
Numero::~Numero(){}//destructor

///OPERADORES///
Numero::operator const char*() const {return numero_.c_str();}

///INCORRECTO///
Numero::Incorrecto::Incorrecto(const Numero::Razon r):r_(r){}
Numero::Razon Numero::Incorrecto::razon() const{return r_;}

///FUNCIONES///
void eliminarChar(Cadena& cad, size_t pos){ //fuera de clase
    Cadena nuevo = cad.substr(0, pos);

    if ((pos + 1) < cad.length())
        nuevo += Cadena(cad.substr(pos+1, cad.length()));

    cad = move(nuevo);
}

/////////////////////////////
///CLASE *TARJETA*///-----
//-----
///CONSTRUCTORES///
Tarjeta::Tarjeta( const Numero& n, Usuario& u, const Fecha& f):numero_(n), usuario_(&u), caducidad_(f), activa_(1), tipo_(esTipo()){
    if (f < Fecha())throw Caducada(f);
    usuario_->es_titular_de(*this);
}

```

```

}

Tarjeta::~Tarjeta(){//destructor
    if(usuario_) usuario_->no_es_titular_de(*this);
}

///OPERADORES///
std::ostream& operator <<(std::ostream& os, const Tarjeta& t)
{
    os << t.tipo() << std::endl << t.numero() << std::endl;
    //Transformar nombre y apellidos en mayúsculas
    Cadena aux = t.titular()->nombre(); //Se asigna nombre a una cadena auxiliar modificable
    int i=0;
    while(aux[i]!='\0')
        {if (islower(aux[i])) aux[i]=toupper(aux[i]);i++;}
    os << aux << " ";

    i=0;
    aux = t.titular()->apellidos(); //Se asigna apellidos a una cadena auxiliar modificable
    while(aux[i]!='\0')
        {if (islower(aux[i])) aux[i]=toupper(aux[i]);i++;}
    os << aux << std::endl;

    //Se formatea la fecha para mostrarla
    os << "Caduca: " << std::setfill ('0') << std::setw (2) << t.caducidad().mes() << "/" << std::setw (2) << (t.caducidad().anno() % 100) << std::endl;
    return os;
}

ostream& operator <<(ostream& os, const Tarjeta::Tipo& tipo){
    switch(tipo){
        case Tarjeta::VISA: os << "VISA"; break;
        case Tarjeta::Mastercard: os << "Mastercard"; break;
        case Tarjeta::Maestro: os << "Maestro"; break;
        case Tarjeta::JCB: os << "JCB"; break;
        case Tarjeta::AmericanExpress: os << "AmericanExpress"; break;
        case Tarjeta::Otro: os << "Otro"; break;
        default: os << "Error";
    }
    return os;
}
bool operator <(const Numero& a, const Numero& b){return strcmp(a, b) < 0;}
bool operator <(const Tarjeta& a, const Tarjeta& b){return a.numero() < b.numero();}
bool operator >(const Tarjeta& a, const Tarjeta& b){return b.numero() < a.numero();}

///OBSERVADORES///

bool& Tarjeta::activa(bool a){activa_=a; return activa_;}

```

///CADUCADA///

```

Tarjeta::Caducada::Caducada(const Fecha& f):fecha_caducada(f){}

```

///NUM_DUPPLICADO///

```

Tarjeta::Num_duplicado::Num_duplicado(const Numero& n):numer_(n){}
const Numero& Tarjeta::Num_duplicado::que()const{return numer_;}

```

///DEACTIVADA///

```

Tarjeta::Desactivada::Desactivada(){}

```

///FUNCIONES///

```

void Tarjeta::anula_titular(){
    activa_=false;
    const_cast<Usuario*&>(usuario_) =nullptr;
}

```

```

Tarjeta::Tipo Tarjeta::esTipo(){
    int a = atoi(numero_.num().substr(0, 2).c_str());
    switch(a/10){
        case 3:
            if(a==34 || a==37) return Tarjeta::AmericanExpress;
            else return Tarjeta::JCB;
            break;
        case 4:
            return Tarjeta::VISA;
            break;
        case 5:
            return Tarjeta::Mastercard;
            break;
        case 6:
            return Tarjeta::Maestro;
            break;
        default:
            return Tarjeta::Otro;
    }
}

```

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



```
usuario.hpp
#ifndef USUARIO_HPP
#define USUARIO_HPP

#include "cadena.hpp"
#include "tarjeta.hpp"
#include "articulo.hpp"
#include <map>
#include <set>
#include <unordered_map>
#include <unordered_set>
#include <unistd.h>

class Numero;
class Tarjeta;
///////////////
//CLASE *CLAVE*-----
//-----
class Clave{
    private:
        //VARIABLES///
        Cadena pass_c;

    public:
        //CONSTRUCTORES///
        Clave(const char*); 
        ~Clave(); //destructor

        //OBSERVADORES///
        Cadena clave() const; //devuelve la contraseña cifrada

        //CONSTANTES///
        enum Razon{CORTA, ERROR_CRYPT};

        //FUNCIONES///
        bool verifica (const char*)const;

        //INCORRECTA///
        class Incorrecta {
            public:
                Incorrecta(const Clave::Razon r);
                Clave::Razon razon() const;
            private:
                Clave::Razon r_;
        };
    protected:
};

///////////////
//CLASE *USUARIO*-----
//-----

class Usuario{
    public:
        //CONSTRUCTORES///
        Usuario(const Cadena& _id, const Cadena& _nom, const Cadena& _apell, const Cadena& _direcc, const Clave& _pass);
        Usuario(const Usuario& that) = delete;
        Usuario& operator=(const Usuario &)=delete;
        ~Usuario(); //destructor

        //CONSTANTES///
        typedef std::map<Numero, Tarjeta*> Tarjetas;
        typedef std::unordered_map<Articulo*,unsigned> Articulos;
```



WUOLAH

```

////ID_DUPPLICADO////
class Id_duplicado{
    public:
        Id_duplicado(const Cadena&);
        const Cadena& idd()const; //observador
    private:
        Cadena idd_;
};

////OBSERVADORES////
const Cadena& id() const noexcept{return ID_;}
const Cadena& nombre() const noexcept{return nombre_;}
const Cadena& apellidos() const noexcept{return apell_;}
const Cadena& direccion() const noexcept{return direcc_;}
const Tarjetas& tarjetas() const noexcept{return tarjetas_;}
const size_t n_articulos()const noexcept{return n_articulos_;}
const Articulos& compra() const noexcept{return articulos_;}

////FUNCIONES////
void es_titular_de(const Tarjeta&);           //se le asocia una tarjeta
void no_es_titular_de(Tarjeta&);               //se desasocia una tarjeta
void compra(const Articulo& a, size_t cant=1);

friend std::ostream& operator << (std::ostream& os, const Usuario &u) noexcept;

private:
    ////VARIABLES///
    Cadena ID_, nombre_, apell_, direcc_;
    Clave pass_;
    Tarjetas tarjetas_;
    Articulos articulos_;
    size_t n_articulos_;
    static std::unordered_set<Cadena> registrados;

protected:
};

std::ostream& operator << (std::ostream& os, const Usuario& u) noexcept;
void mostrar_carro (std::ostream& os,const Usuario& u);
#endif

```

```

usuario.cpp
#include "usuario.hpp" // CONTIENE CLASE *USUARIO* Y CLASE *CLAVE*
#include "tarjeta.hpp"
#include "articulo.hpp"

#include <unistd.h>
#include <cstdlib>
#include <string.h>
#include <random>
#include <iomanip>
#include <set>

using namespace std;
unordered_set<Cadena> Usuario::registrados;
///////////////////////////////
///CLASE *CLAVE*///-----
//-----
///CONSTRUCTOR///
Clave::Clave(const char* p){
    setlocale(LC_ALL, "");
    if(strlen(p)<5) throw Clave::Incorrecta(Clave::CORITA); //ERROR CORTA

    random_device r;
    uniform_int_distribution<size_t> dist(0, 63);
    char const MD5chars[] = "./ABCDEFHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
    char salt[2] = {MD5chars[dist(r)],MD5chars[dist(r)]};

    if(crypt(p, salt) == nullptr) throw Clave::Incorrecta(Clave::ERROR_CRYPT); //ERROR ERROR_CRYPT
    pass_c = crypt(p, salt);
}
Clave::~Clave(){           //destructor
}

///INCORRECTA///
Clave::Incorrecta::Incorrecta(const Clave::Razon r):r_(r){
}
Clave::Razon Clave::Incorrecta::razon() const {return r_;}

///FUNCIONES///
bool Clave::verifica(const char* c) const {
    char * p=crypt(c, pass_c.c_str());
    return p == pass_c;
}

///OBSERVADORES///
Cadena Clave::clave() const {return pass_c;}

/////////////////////////////
///CLASE *USUARIO*///-----
//-----
///CONSTRUCTORES///
Usuario::Usuario(const Cadena& _id, const Cadena& _nomb, const Cadena& _apell, const Cadena& _direcc, const Clave& _pass)
:_ID(_id), nombre(_nomb), apell(_apell), direcc(_direcc), pass(_pass), n_articulos_(articulos_.size()){
    if(!registrados.insert(ID_).second)
        throw Id_duplicado(ID_);
}
Usuario::~Usuario(){
    for(auto& i : tarjetas_)
        i.second->anula_titular();
    registrados.erase(ID_); //borra el id del usuario
}

///ID DUPLICADO///
Usuario::Id_duplicado::Id_duplicado(const Cadena& c):idd_(c){}
const Cadena& Usuario::Id_duplicado::idd() const {return idd_;} //observador

```

```

///FUNCIONES///
void Usuario::es_titular_de(const Tarjeta &t){
    if(t.titular() == this) tarjetas_[t.numero()] = const_cast<Tarjeta*>(&t);
}
void Usuario::no_es_titular_de(Tarjeta &t){tarjetas_.erase(t.numero());}

void Usuario::compra(const Articulo& a, size_t cant){
    Usuario::Articulos::const_iterator got = articulos_.find (const_cast<Articulo*>(&a));

    if (got == articulos_.end() ){
        if(cant > 0){
            articulos_[const_cast<Articulo*>(&a)] = cant;
            n_articulos_++;
        }
    }else{
        if(cant > 0 ){
            articulos_[const_cast<Articulo*>(&a)] = cant;
        }else{
            articulos_.erase(const_cast<Articulo*>(&a));
            n_articulos--;
        }
    }
}

```

///OPERADORES///

```

std::ostream& operator << (std::ostream& os, const Usuario& u) noexcept{
    setlocale(LC_ALL, "");
    os<<u.ID_<<"["<<u.pass_clave()<<"] "<<u.nombre_<<" "<<u.apell_<<endl;
    os<<u.direcc_<<endl;
    os<<"Tarjetas:\n";
    for (auto& t : u.tarjetas_)
        os << *t.second << endl;
    return os;
}
void mostrar_carro (ostream& os, const Usuario& u){

    os<<"Carrito de compra de "<<u.id()<<" [Artículos: "<<u.n_articulos()<<"]"<< endl<< " Cant.\tArtículo" <<endl;
    os <<"======"<< endl;

    for(Usuario::Articulos::const_iterator i = u.compra().begin(); i != u.compra().end(); i++){
        os << " " << i->second << "\t" << "[" << i->first->referencia() << "] \" " << i->first->titulo() << "\", ";
        os << i->first->f_publi().anno() << ". " << setprecision(2) << std::fixed << i->first->precio() << " €" << endl;
    }
}

```

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



```
pedido-articulo.hpp
#ifndef Pedido_Articulo_HPP_
#define Pedido_Articulo_HPP_

#include<iostream>
#include <iomanip>
#include "articulo.hpp"
#include "pedido.hpp"

///////////////////////////////
///CLASE *LineaPedido*///-----
//-----
class LineaPedido{
public:
    ///CONSTRUCTOR///
    explicit LineaPedido(double pr, unsigned c=1);

    ///OBSERVADORES///
    const double precio_venta() const noexcept {return precio_;}
    const unsigned cantidad() const noexcept {return cantidad_;}

private:
    ///VARIABLES///
    double precio_;
    unsigned cantidad_;

protected:
};

std::ostream& operator<<(std::ostream& os, const LineaPedido& p);

class Pedido;
class Articulo;
///FUNCIONES DE CLASE///
class OrdenaPedidos: public std::binary_function<Pedido*, Pedido*, bool>{
public:
    bool operator()(const Pedido* p1, const Pedido* p2) const;
};

class OrdenaArticulos: public std::binary_function<Articulo*, Articulo*, bool>{
public:
    bool operator()(const Articulo* a1, const Articulo* a2) const {return (a1->referencia() < a2->referencia());}
};

///////////////////////////////
///CLASE *Pedido_Articulo*///-----
//-----
class Pedido_Articulo{
public:
    ///LIBRERIAS///
    typedef std::map<Articulo*, LineaPedido, OrdenaArticulos> ItemsPedido;
    typedef std::map<Pedido*, LineaPedido, OrdenaPedidos> Pedidos;
    typedef std::map<Pedido*, ItemsPedido, OrdenaPedidos> PedArt;
    typedef std::map<Articulo*, Pedidos, OrdenaArticulos> ArtPed;

    ///OBSERVADORES///
    //PEDIR//
    void pedir(Pedido& ped, Articulo& art, double precio, unsigned cantidad=1); //PEDIDO - ARTICULO
    void pedir(Articulo& art, Pedido& ped, double precio, unsigned cantidad=1); //ARTICULO - PEDIDO
    //DETALLE//
    const ItemsPedido& detalle(Pedido& p);
    //VENTAS//
    Pedidos ventas(Articulo& a);
    //mostrarDetallePedidos//
    std::ostream& mostrarDetallePedidos(std::ostream& os) const;
    //mostrarVentasArticulos//
    std::ostream& mostrarVentasArticulos(std::ostream& os) const;
```



WUOLAH

```
private:  
    ///////////////////////////////////////////////////////////////////  
    PedArt ped_art_;  
    ArtPed art_ped_;  
  
protected:  
};  
  
std::ostream& operator <<(std::ostream& os,const Pedido_Articulo::ItemsPedido& ip);  
std::ostream& operator <<(std::ostream& os,const Pedido_Articulo::Pedidos& p);  
  
#endif
```

```

pedido-articulo.cpp
#include "pedido-articulo.hpp"
using namespace std;

///////////////////////////////
///CLASE *LineaPedido*///-----
//-----
///CONSTRUCTOR///
LineaPedido::LineaPedido(double pr, unsigned c)
: precio_(pr), cantidad_(c){}

///OPERATOR <<///
std::ostream& operator<<(std::ostream& os, const LineaPedido& p){

    os<<fixed<<setprecision(2)<<p.precio_venta()<<" €\t"<<p.cantidad();
    return os;
}

///////////////////////////////
///CLASE *Pedido_Articulo*///-----
//-----
bool OrdenaPedidos::operator()(const Pedido* p1,const Pedido* p2) const{return (p1->numero() < p2->numero());}

//PEDIR//
void Pedido_Articulo::pedir (Pedido& ped, Articulo& art, double precio, unsigned cantidad ){
    ped_art_[&ped].insert(make_pair(&art,LineaPedido(precio,cantidad)));
    art_ped_[&art].insert(make_pair(&ped,LineaPedido(precio,cantidad)));
}
void Pedido_Articulo::pedir (Articulo&a, Pedido&p, double precio, unsigned cantidad ){
    pedir(p,a,precio,cantidad);
}
//DETALLE//
const Pedido_Articulo::ItemsPedido& Pedido_Articulo::detalle(Pedido&p){
    return ped_art_.find(&p)->second;
}
//VENTAS//
Pedido_Articulo::Pedidos Pedido_Articulo::ventas(Articulo&a){
    if(art_ped_.find(&a)!=art_ped_.end()){
        return art_ped_.find(&a)->second;
    }else{
        Pedido_Articulo::Pedidos vacio;
        return vacio;
    }
}
//mostrarDetallePedidos//
ostream& Pedido_Articulo::mostrarDetallePedidos(ostream&os) const{
    double precio = 0.0;

    for(auto& iter : ped_art_){
        os << "Pedido núm. " << iter.first->numero();
        os << "\tCliente: " << iter.first->tarjeta()->titular()->nombre() << "\n"; //antes estaba separado el espacio y el \n
        os << "Fecha: " << iter.first->fecha() << iter.second;
        precio += iter.first->total();
    }
    os << "TOTAL VENTAS: " << fixed<<setprecision(2) << precio << " €" << endl;
    return os;
}

//mostrarVentasArticulos//
ostream& Pedido_Articulo::mostrarVentasArticulos(ostream&os) const{
    for(auto& iter: art_ped_){
        os << "Ventas de " << "[" << iter.first->referencia() << "]";
        os << "\t" << iter.first->titulo() << "\n" << iter.second << endl;
    }
    return os;
}

```

```

///OPERADORES <>///
ostream& operator <<(ostream&os,const Pedido_Articulo::ItemsPedido& ip)
{
    double precio = 0;
    Pedido_Articulo::ItemsPedido::const_iterator i;
    os << endl << "=====\\n" << endl;
    os << " PVP \t Cant. \t Articulo \\n";
    os << "=====\\n" << endl;
    for(i = ip.begin(); i != ip.end(); ++i){
        os << " " << i->second.precio_venta() << "€\\t";
        os << i->second.cantidad() << "\\t";
        os << "[" << i->first->referencia() << "] ";
        os << "\"" << i->first->título() << "\"" << endl;
        precio = precio + i->second.precio_venta() * i->second.cantidad();
    }
    os << "=====\\n" << endl;
    os << fixed;
    os << setprecision(2) << precio << " €" << endl;
    return os;
}
ostream& operator <<(ostream&os,const Pedido_Articulo::Pedidos& pedidos)
{
    double precio = 0;
    unsigned total = 0;
    Pedido_Articulo::Pedidos::const_iterator i;
    os << "=====\\n" << endl;
    os << " PVP \t Cant. \t Fecha venta \\n";
    os << "=====\\n" << endl;
    for(auto iter : pedidos){
        os << " " << iter.second.precio_venta() << "€\\t";
        os << iter.second.cantidad() << "\\t";
        os << iter.first->fecha() << endl;
        precio += (iter.second.precio_venta() * iter.second.cantidad());
        total += iter.second.cantidad();
    }
    os << "=====\\n" << endl;
    os << fixed;
    os << setprecision(2) << precio << " €\\t" << total << endl ;
    return os;
}

```

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



```
usuario-pedido.hpp
#ifndef USUARIO_PEDIDO_HPP_
#define USUARIO_PEDIDO_HPP_

#include <map>
#include <set>

class Pedido;
class Usuario;

class Usuario_Pedido{
public:
    //DEFINICIONES///
    typedef std::set<Pedido*> Pedidos;
    typedef std::map<Usuario*, Pedidos> UsuarioP;
    typedef std::map<Pedido*, Usuario*> PedidoU;

    //OBSERVADORES///
    //ASOCIA//
    void asocia(Usuario& us, Pedido& ped){usu_ped_[&us].insert(&ped); ped_usu_[&ped] = &us;}
    void asocia(Pedido& ped, Usuario& us){asocia(us, ped);}

    //PEDIDOS//
    Pedidos& pedidos(Usuario& us){return usu_ped_.find(&us)->second;}
    //CLIENTE//
    Usuario* cliente(Pedido& ped){return ped_usu_.find(&ped)->second;}

private:
    //VARIABLES///
    UsuarioP usu_ped_;
    PedidoU ped_usu_;
protected:
};

#endif

usuario-pedido.cpp
#include "usuario-pedido.hpp"
```

