may 01, 18 17:41 **Thread.h** Page 1/1

```
1  #ifndef THREAD_H
2  #define THREAD_H
3  #include <iostream>
4  #include <vector>
5  #include <thread>
6  #include <mutex>
7
8  class Thread {
9      private:
10         std::thread thread;
11     public:
12         /*Constructor*/
13         Thread();
14     /*Crea e inicializa el thread con functor run*/
15     void start();
16         /*Hace el  join del thread*/
17         void join();
18         /*Metodo run a ser implementado si se hereda o implementa la clase*/
19         virtual void run() = 0;
20         /*Destructor*/
21         virtual ~Thread();
22         /*Constructor por copia no permitido*/
23         Thread(const Thread&) = delete;
24         /*Operador = no permitido*/
25         Thread& operator=(const Thread&) = delete;
26         /*Constructor por movimiento*/
27         Thread(Thread∧ other);
28         /*Operador = no permitido*/
29         Thread& operator=(Thread∧ other);
30 };
31 #endif
```

may 01, 18 17:41 **Thread.cpp** Page 1/1

```
1  #include "Thread.h"
2  #include <iostream>
3  #include <vector>
4  #include <thread>
5  #include <mutex>
6
7
8  Thread::Thread(){
9  }
10
11 void Thread::start(){
12     thread = std::thread(&Thread::run, this);
13 }
14
15 void Thread::join(){
16     thread.join();
17 }
18
19 Thread::~Thread(){
20 }
21
22 Thread::Thread(Thread∧ other){
23     this→thread = std::move(other.thread);
24 }
25
26 Thread& Thread::operator=(Thread∧ other){
27     this→thread = std::move(other.thread);
28     return *this;
29 }
```

may 01, 18 17:41 **Sinc_class.h** Page 1/1

```
1  #ifndef SINC_CLASS_H
2  #define SINC_CLASS_H
3  #include <iostream>
4  #include <vector>
5  #include <thread>
6  #include <mutex>
7
8  /*Clase para realizar la sincronizacion de metodos, para que no se
9  produzca race condition*/
10
11 class Sinc_class {
12     private:
13         std::mutex &m;
14         /*Constructor por copia no permitido*/
15         Sinc_class(const Sinc_class&) = delete;
16         /*Operador = no permitido*/
17         Sinc_class& operator=(const Sinc_class&) = delete;
18         /*Constructor por movimiento no permitido*/
19         Sinc_class(Sinc_class∧) = delete;
20         /*Operador = no permitido*/
21         Sinc_class& operator=(Sinc_class∧) = delete;
22     public:
23         /*Constructor que crea un mutex en memoria*/
24         explicit Sinc_class(std::mutex &m);
25         /*Destructor que destruye en memoria el mutex almacenado*/
26         ~Sinc_class();
27 };
28 #endif
29
```

may 01, 18 17:41 **Sinc_class.cpp** Page 1/1

```
1  #include <iostream>
2  #include <vector>
3  #include <thread>
4  #include <mutex>
5  #include "Sinc_class.h"
6
7  Sinc_class::Sinc_class(std::mutex &m) : m(m) {
8      m.lock();
9  }
10
11 Sinc_class::~Sinc_class() {
12     m.unlock();
13 }
```

may 01, 18 17:41 **Paquete.h** Page 1/1

```
#ifndef PAQUETE_H
#define PAQUETE_H
#include <iostream>
#include <vector>
#include <fstream>
#include <string>

class Paquete{
    private:
        int tipo;
        std::string tipo_n;
        int cant;
        int lim_paq;
        std::vector<int> vect_anchos;

    public:
        /*Constructor: define la estructura del paquete segun el tipo
        nombre y limite de carga en paquete*/
        Paquete(int tipo,const std::string& tipo_n,int lim_paq);
        /*Devuelve la cantidad que no pudo ser agregada, devuelve 0 si
        pudo agregar todo o -1 en caso que esta lleno*/
        int agregar(int cant,int ancho);
        /*Devuelve la cantidad de tornillos que aun se pueden almacenar*/
        int cant_a_comp() const;
        /*Devuelve la cantidad de tornillos almacenados*/
        int obt_cant() const;
        /*Devuelve el tipo en int segun el archivo clasificador*/
        int obt_tipo() const;
        /*Devuelve la mediana de los anchos de los tonrillos del paquete*/
        int obt_mediana() const;
        /*Devuelve el limite de carga del paquete*/
        int obt_lim() const;
        /*Devuelve un string con el nombre del paquete*/
        const std::string& obt_nombre() const;
        /*Operador menor, devuelve un booleano si es menor a otro Paquete*/
        bool operator < (const Paquete& a) const;
};


#endif

```

may 01, 18 17:41 **Paquete.cpp** Page 1/2

```
#include "Paquete.h"
#include <algorithm>
#include <string>
#include <vector>
#define CERO 0
#define DOS 2

Paquete::Paquete(int tipo,const std::string& tipo_n,int lim_paq):
  tipo(tipo),tipo_n(tipo_n),lim_paq(lim_paq){
  this→cant = CERO;
}

int Paquete::obt_cant() const{
  return cant;
}

int Paquete::obt_tipo() const{
  return tipo;
}

const std::string& Paquete::obt_nombre() const{
  return tipo_n;
}

int Paquete::obt_lim() const{
  return lim_paq;
}

int Paquete::obt_mediana() const{
  std::vector<int> auxiliar(vect_anchos); //para asegurar el const
  unsigned int aux = auxiliar.size();
  sort(auxiliar.begin(), auxiliar.end());
  if(aux%DOS≡CERO){//par
    return ((auxiliar[(aux/DOS)-1] + auxiliar[aux/DOS])/DOS);
  }
  return  auxiliar[aux/DOS];
}

int Paquete::agregar(int cant,int ancho){
  if(this→cant_a_comp()≡CERO){
    return -1;
  }
  if(this→cant_a_comp() ≤ cant){
    int aux = cant-this→cant_a_comp();
    for (int i = CERO; i < this→cant_a_comp(); i++){
      vect_anchos.push_back(ancho);
    }

    this→cant = this→lim_paq;
    std::cout << "Paquete listo: "<<this→obt_cant()<<" tornillos";
    std::cout << " de tipo "<<this→obt_nombre();
    std::cout << " (mediana: "<<this→obt_mediana()<<")"<<'\n';

    return  aux;
  }
  this→cant += cant;
  for (int i = CERO; i < cant; i++){
    vect_anchos.push_back(ancho);
  }
  return CERO;
}

int Paquete::cant_a_comp() const{
  return (this→lim_paq - this→cant);
}

```

may 01, 18 17:41 **Paquete.cpp** Page 2/2

```
67  bool Paquete::operator < (const Paquete& a) const{
68      return (tipo < (a.obt_tipo()));
69  }
70
71
72
73
74
```

may 01, 18 17:41 **main.cpp** Page 1/1

```
1   #include "clasificador.h"
2   #include "Empaquetador.h"
3   #include "exc_file_err.h"
4   #include <vector>
5   #define ARCH_CONFIG 1
6   #define ARCHS_CLASIF 2
7   #define CERO 0
8
9   int main(int argc, char *argv[]){
10    Empaquetador empaq(argv[ARCH_CONFIG]);
11    std::vector<Thread*> threads;
12    for (int i = ARCHS_CLASIF; i < argc; i++) {
13          try{
14            threads.push_back(new Clasificador(argv[i],empaq));
15          }
16          catch(FileError& ex){
17            std::cerr << ex.what() << std::endl;
18          }
19          catch(const std::exception &e) {
20      std::cerr << e.what() << std::endl;
21      }
22      }
23      for (unsigned int i = CERO; i < threads.size(); i++){
24          threads[i]→start();
25      }
26      for (unsigned int i = CERO; i < threads.size(); i++){
27          threads[i]→join();
28          delete threads[i];
29      }
30      empaq.ordenar_paquete();
31    empaq.imp_restantes();
32    return CERO;
33  }
```

may 01, 18 17:41 **exc_file_err.h** Page 1/1

```
#ifndef FILE_ERR_EXCEP_H
#define FILE_ERR_EXCEP_H
#include <exception>
#include <string>

class FileError: public std::exception{
  private:
      std::string msg;
  public:
    /*Defino el constructor de la excepcion*/
    explicit FileError(const std::string& msg);
    /*Defino el metodo what para mostrar el mensaje*/
    virtual const char* what() const throw();
};

#endif
```

may 01, 18 17:41 **exc_file_err.cpp** Page 1/1

```
#include "exc_file_err.h"
#include <string>


FileError::FileError(const std::string& msg): msg(msg){}

const char* FileError::what() const throw() {
  return msg.c_str();
}

```

may 01, 18 17:41 **Empaquetador.h** Page 1/1

```
#ifndef EMPAQUETADOR_H
#define EMPAQUETADOR_H
#include <iostream>
#include <vector>
#include <fstream>
#include <thread>
#include <mutex>
#include "Paquete.h"
#include <string>

class Empaquetador{
  public:
    /*Ordena los paquetes*/
    void ordenar_paquete();
    /*Constructor por copia no permitido*/
    Empaquetador(const Empaquetador& other)=delete;
    /*Operador = no permitido*/
    Empaquetador& operator=(const Empaquetador& other)=delete;
    /*Constructor: intenta cargar el archivo configuracion y
    segun lo leido armar los paquetes, si no puede cargar el archivo
    lanza una excepcion del tipo FileError*/
    explicit Empaquetador(const std::string& nomb_file);
    /*Constructor por movimiento*/
    Empaquetador(Empaquetador∧ other);
    /*Destructor: se elimina de memoria los paquetes*/
    ~Empaquetador();
    /*Carga en el paquete la cantidad pasada y ancho segun el tipo*/
    void carg_armar_empaq(int tipo, int cantidad, int ancho);
    /*Imprime los paquetes segun su tipo*/
    void imp_restantes() const;
    //Fin public
    private:
      /*Atributos privados*/
    std::string nomb_file;
    std::ifstream myfile;
    std::vector<Paquete*> vect_paqs;
    std::mutex m;
    /*Devuelve −1 si no pudo agregar al paquete, 0 si pudo todo y sino
    devuelve la cantidad que no pudo agregar*/
    int agregar_paq(int tipo,int cantidad,int ancho);
    /*Revisa si hay algun paquete completado para quitarlo y agregar uno
    del mismo tipo vacio*/
    void revisar_completo(int tipo);
};
#endif
```

may 01, 18 17:41 **Empaquetador.cpp** Page 1/2

```
#include "Empaquetador.h"
#include "Sinc_class.h"
#include <algorithm>
#include <string>
#define CERO 0
#define INVAL −1
#define DESPLAZAMIENTO 1
#include "exc_file_err.h"


Empaquetador::Empaquetador(const std::string& nomb_file): nomb_file(nomb_file){
  std::string nombre;
  std::string id;
  std::string lim;
  unsigned int pos = CERO;
  myfile.open(nomb_file,std::ifstream::in);
  if(¬myfile.is_open()){
    std::string msg=":no se pudo cargar la configuracion del Empaquetador";
    // no va a estar mal la config ni tampoco los archivos de los clasif
    throw FileError(nomb_file+msg);
  }
  std::string line;
  while (std::getline(myfile, line)){
    pos = CERO;
    for (unsigned int i = CERO; i < line.size(); i++){
      if((line[i]≡'=') ∧ (i≠CERO)){
        id=line.substr(CERO,i−pos);
        pos=i+DESPLAZAMIENTO;
      }
      if( (line[i]≡',') ∧ (pos≠CERO) ∧ (pos≠i) ){
        nombre=line.substr(pos,i−pos);
        lim=line.substr(i+DESPLAZAMIENTO,line.size());
        break;
      }
    }
    int id_aux =std::stoi(id);
    int lim_aux = std::stoi(lim);
    Paquete *paq = new Paquete(id_aux,nombre,lim_aux);
    vect_paqs.push_back(paq);
  }
}


Empaquetador::Empaquetador(Empaquetador∧ other): nomb_file(other.nomb_file){
  this→vect_paqs = other.vect_paqs;
  this→myfile = std::move(other.myfile);
}

int Empaquetador::agregar_paq(int tipo,int cantidad,int ancho){
  Sinc_class sincronizar(m);
  int pos=INVAL;
  for (unsigned int i = CERO; i < vect_paqs.size(); i++){
    if(vect_paqs[i]→obt_tipo() ≡ tipo){
      pos = i;
      break;
    }
  }
  if(pos≡INVAL){
    std::cerr<<"Tipo de tornillo invalido: "<<tipo<<'\n';
    return CERO;
  }
  return (vect_paqs[pos]→agregar(cantidad,ancho));;
}

void Empaquetador::revisar_completo(int tipo){
  Sinc_class sincronizar(m);
```

```
67    for (unsigned int i = CERO; i < vect_paqs.size(); i++){
68      if(vect_paqs[i]→cant_a_comp() ≡ CERO){
69        Paquete *paq_nuevo = new Paquete(vect_paqs[i]→obt_tipo(),
70                    vect_paqs[i]→obt_nombre(),
71                    vect_paqs[i]→obt_lim());
72        delete vect_paqs[i];
73        vect_paqs.erase(vect_paqs.begin() + i); //borre el que se completo
74        vect_paqs.push_back(paq_nuevo); //agrego el nuevo
75      }
76    }
77  }
78
79  void Empaquetador::carg_armar_empaq(int tipo, int cantidad, int ancho){
80    int aux=INVAL;
81    while(aux≡INVAL){
82      aux = agregar_paq(tipo,cantidad,ancho);
83      this→revisar_completo(tipo);
84      if(aux>CERO){
85        this→carg_armar_empaq(tipo,aux,ancho);
86      }
87    }
88  }
89
90
91
92  struct {
93    bool operator()(Paquete* a, Paquete* b){
94      return (*a < *b);
95    }
96  }paq_orden;
97
98  void Empaquetador::ordenar_paquete(){
99    std::sort(vect_paqs.begin(), vect_paqs.end(),paq_orden);
100 }
101
102 void Empaquetador::imp_restantes() const{
103   std::cout<<"# Informe de remanentes"<<'\n';
104   for (unsigned int i = CERO; i < vect_paqs.size() ; i++){
105     std::cout<<"* "<< vect_paqs[i]→obt_cant() <<" tornillos";
106     std::cout<<" de tipo "<< vect_paqs[i]→obt_nombre()<<'\n';
107   }
108 }
109
110
111 Empaquetador::~Empaquetador(){
112   for (unsigned int i = CERO; i < vect_paqs.size(); i++){
113     delete vect_paqs[i];
114   }
115   if(myfile.is_open()){
116     myfile.close();
117   }
118 }
119
```



```
1   #ifndef CLASIFICADOR_H
2   #define CLASIFICADOR_H
3   #include <string>
4   #include <fstream>
5   #include "Empaquetador.h"
6   #include "Thread.h"
7   #include <mutex>
8
9
10  class Clasificador : public Thread {
11    private:
12      std::string nomb_clf;
13      std::string nomb_file;
14      std::ifstream myfile;
15      Empaquetador &empaq;
16      std::mutex m;
17    public:
18      /*Constructor por copia no permitido*/
19      Clasificador(const Clasificador& other)=delete;
20      /*Operador = no permitido*/
21      Clasificador& operator=(const Clasificador& other)=delete;
22      /*Constructor: intenta abrir el archivo nomb_file y obtener el
23      nombre del clasificador, lanza excepcion del tipo FIleError
24       si no puede abrir el archivo*/
25      Clasificador(const std::string& nomb_file, Empaquetador &empaq);
26      /*Constructor por movimietno*/
27      Clasificador(Clasificador∧ other);
28      /*Re define la funcion run de Thread, donde se realiza la lectura
29      del archivo si es que fue abierto con exito y enpaqueta lo
30      leido del archivo*/
31      virtual void run() override;
32  };
33  #endif
```

```
#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>
#include <bitset>
#define SIZE_BUFF 4
#define CERO_H 0x00
#define V_ATASCADO 0xFFFFFFFF
#define BITS_27 27
#define BITS_5 5
#define BITS_10 10
#include "clasificador.h"
#include "Empaquetador.h"
#include <arpa/inet.h>
#include "Sinc_class.h"
#include "exc_file_err.h"

Clasificador::Clasificador(const std::string& nomb_file, Empaquetador &empaq):
  nomb_file(nomb_file), empaq(empaq) {
  myfile.open(nomb_file, std::ifstream::in);
  if(¬myfile.is_open()){
    throw FileError((nomb_file+": no se pudo conectar con el dispositivo"));
  }else{
    while(true){
        char buff;
          myfile.get(buff);
      if(myfile.eof()){
        throw FileError((nomb_file+": no se pudo conectar con el dispositivo"));
      }
      if(buff≡CERO_H) {
        break;
      }
      nomb_clf.push_back(buff);
    }
    std::cout<<nomb_file;
    std::cout<<": se establece conexion con el dispositivo ";
    std::cout<<nomb_clf<<'\n';
  }
  }

Clasificador::Clasificador(Clasificador∧ other):
  nomb_file(other.nomb_file), empaq(other.empaq) {
  this→nomb_clf = other.nomb_clf;
  this→nomb_file = other.nomb_file;
  this→myfile = std::move(other.myfile);
  other.nomb_file = nullptr;
  other.nomb_clf = nullptr;
}



void Clasificador::run(){
  unsigned int tipo_tornillo;
  unsigned int ancho;
  unsigned int cantidad;
  while(¬myfile.eof()){
    uint32_t little_endian;
    uint32_t big_endian;
    myfile.read((char*)&big_endian, SIZE_BUFF);
    if(myfile.eof()){ //se debe a que repite la ultima lectura...
      break;
    }
    little_endian = ntohl(big_endian);
    if(little_endian≡(V_ATASCADO)){
      Sinc_class sincronizar(m);
      std::cerr << nomb_clf << " atascado"<<'\n';
```



```
      continue;
    }


    tipo_tornillo = little_endian>>BITS_27;
    ancho = (little_endian<<BITS_27)>>BITS_27;
    cantidad = (little_endian<<BITS_5)>>BITS_10;
    empaq.carg_armar_empaq(tipo_tornillo,cantidad,ancho);
    }
  }

```

may 01, 18 17:41 **Table of Content** Page 1/1

Table of Contents