```
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #ifndef TP_USERCOMMAND_H
6  #define TP_USERCOMMAND_H
7
8  #include "server_Command.h"
9  #include "server_Login.h"
10 #include <map>
11 #include <string>
12
13 class UserCommand : public Command {
14 private:
15     std::string user;
16     std::map<std::string,std::string> &cfg;
17     Login& login;
18 public:
19     UserCommand(std::string user,
20             std::map<std::string,std::string> &cfg,
21             Login& login);
22     std::string execute() override;
23     ~UserCommand() override;
24 };
25
26 #endif //TP_USERCOMMAND_H
```

```
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #include "server_UserCommand.h"
6
7  #define USER_KEY "passRequired"
8  #define CODE "331 "
9
10 UserCommand::UserCommand(std::string user,
11         std::map<std::string,std::string> &cfg, Login& login) :
12     user(std::move(user)),
13     cfg(cfg),
14     login(login) {}
15
16 std::string UserCommand::execute() {
17     login.enterUser(user);
18     return CODE + cfg.find(USER_KEY)→second;
19 }
20
21 UserCommand::~UserCommand() {}
```

```cpp
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #ifndef TP_UNKNOWNCOMMAND_H
6  #define TP_UNKNOWNCOMMAND_H
7
8  #include <string>
9  #include <map>
10 #include "server_Command.h"
11
12 class UnknownCommand : public Command {
13 private:
14     std::map<std::string, std::string> &cfg;
15     Login& login;
16 public:
17     UnknownCommand(std::map<std::string, std::string> &cfg, Login& login);
18     std::string execute() override;
19     ~UnknownCommand() override;
20 };
21
22
23 #endif //TP_UNKNOWNCOMMAND_H
```

```cpp
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #include "server_UnknownCommand.h"
6
7  #define UNKNOWN_COMMAND_KEY "unknownCommand"
8  #define UNLOGGED_KEY "clientNotLogged"
9  #define CODE   "530 "
10
11 UnknownCommand::UnknownCommand(std::map<std::string,
12         std::string> &cfg, Login& login) :
13     cfg(cfg),
14     login(login) {}
15
16 std::string UnknownCommand::execute() {
17     login.resetIfNotLogged();
18     if (login.userIsLogged()) {
19         return CODE + cfg.find(UNKNOWN_COMMAND_KEY)→second;
20     } else {
21         return CODE + cfg.find(UNLOGGED_KEY)→second;
22     }
23 }
24
25 UnknownCommand::~UnknownCommand() {}
```

```
1  //
2  // Created by leobellaera on 19/9/19.
3  //
4
5  #ifndef FRAME_OF_REFERENCE_THREAD_H
6  #define FRAME_OF_REFERENCE_THREAD_H
7
8  #include <thread>
9
10 class Thread {
11 private:
12     std::thread thread;
13 public:
14     Thread();
15     void start();
16     void join();
17     virtual void run() = 0;
18     Thread(const Thread&) = delete;
19     Thread& operator=(const Thread&) = delete;
20     Thread(Thread& other);
21     Thread& operator=(Thread& other);
22     virtual ~Thread();
23 };
24
25 #endif //FRAME_OF_REFERENCE_THREAD_H
```

```
1  //
2  // Created by leobellaera on 19/9/19.
3  //
4
5  #include "server_Thread.h"
6
7  Thread::Thread() {}
8
9  void Thread::start() {
10     thread = std::thread(&Thread::run, this);
11 }
12
13 void Thread::join() {
14     thread.join();
15 }
16
17 Thread::Thread(Thread& other) {
18     this→thread = std::move(other.thread);
19 }
20
21 Thread& Thread::operator=(Thread& other) {
22     this→thread = std::move(other.thread);
23     return *this;
24 }
25
26 Thread::~Thread() {}
```

```cpp
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #ifndef TP_THCLIENT_H
6  #define TP_THCLIENT_H
7
8  #include "common_Socket.h"
9  #include "server_Thread.h"
10 #include "server_ServerProxy.h"
11 #include "server_DirectoryOrganizer.h"
12 #include "server_Login.h"
13 #include <string>
14 #include <map>
15 #include <atomic>
16
17 class ThClient : public Thread {
18 private:
19     std::map<std::string,std::string> &cfg;
20     Login login;
21     DirectoryOrganizer& dir_organizer;
22     ServerProxy proxy;
23     std::atomic<bool> finished;
24     void executeCommand(std::string& input);
25     void sendWelcomeMsgToClient();
26 public:
27     ThClient(Socket skt, std::map<std::string,std::string>& cfg,
28              DirectoryOrganizer& dir_org);
29     void run() override;
30     void stop();
31     bool isAlive();
32     ~ThClient() override; //override??
33 };
34
35 #endif //TP_THCLIENT_H
```

```cpp
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #include "server_ThClient.h"
6  #include "server_Command.h"
7  #include "common_SocketException.h"
8
9  #define QUIT_COMMAND "QUIT"
10 #define NEW_CLIENT_CODE "220 "
11 #define NEW_CLIENT_KEY "newClient"
12
13 ThClient::ThClient(Socket skt,
14         std::map<std::string,std::string> &cfg,
15         DirectoryOrganizer& dir_org) :
16     cfg(cfg),
17     login(cfg),
18     dir_organizer(dir_org),
19     proxy(std::move(skt)),
20     finished(false) {}
21
22 void ThClient::run() {
23     std::string input;
24     this→sendWelcomeMsgToClient();
25     while (¬finished) {
26         input.clear();
27         try {
28             proxy.receiveClientCommand(input);
29             this→executeCommand(input);
30         } catch (const SocketException& e) {
31             finished = true;
32             return;
33         }
34         if (input ≡ QUIT_COMMAND) {
35             finished = true;
36         }
37     }
38 }
39
40 void ThClient::sendWelcomeMsgToClient() {
41     try {
42         std::string msg = NEW_CLIENT_CODE + cfg.find(NEW_CLIENT_KEY)→second;
43         proxy.sendMsgToClient(msg);
44     } catch (const SocketException& e) {
45         finished = true;
46     }
47 }
48
49 void ThClient::executeCommand(std::string& input) {
50     Command* command = Command::make_command(cfg, input, login, dir_organizer);
51     std::string answer = command→execute();
52     proxy.sendMsgToClient(answer);
53     delete command;
54 }
55
56 bool ThClient::isAlive() {
57     return ¬finished;
58 }
59
60 void ThClient::stop() {
61     proxy.stopCommunication();
62 }
63
64 ThClient::~ThClient() {}
```

```
1   //
2   // Created by leobellaera on 28/9/19.
3   //
4
5   #ifndef TP_THACCEPTOR_H
6   #define TP_THACCEPTOR_H
7
8   #include <string>
9   #include <map>
10  #include <vector>
11  #include "server_AcceptorSocket.h"
12  #include "server_DirectoryOrganizer.h"
13  #include "server_ThClient.h"
14  #include "server_Thread.h"
15
16  class ThAcceptor : public Thread {
17  private:
18      AcceptorSocket acceptor_skt;
19      DirectoryOrganizer& dir_organizer;
20      std::map<std::string, std::string>& cfg;
21      std::vector<ThClient*> clients;
22      void deleteDeadClients();
23  public:
24      ThAcceptor(DirectoryOrganizer& dir_organizer,
25              std::map<std::string, std::string>& cfg,
26              const char* service, int backlog);
27      void run() override;
28      void stop();
29      ~ThAcceptor() override;
30  };
31
32  #endif //TP_THACCEPTOR_H
```

```
1   //
2   // Created by leobellaera on 28/9/19.
3   //
4
5   #include "server_ThAcceptor.h"
6   #include "common_SocketException.h"
7   #include "server_ThClient.h"
8   #include <iostream>
9
10  ThAcceptor::ThAcceptor(DirectoryOrganizer& dir_organizer,
11          std::map<std::string, std::string>& cfg,
12          const char* service, int backlog) :
13      acceptor_skt(backlog, service),
14      dir_organizer(dir_organizer),
15      cfg(cfg) {}
16
17
18  void ThAcceptor::run() {
19      while (true) {
20          try {
21              Socket skt = acceptor_skt.accept();
22              ThClient* thclient = new ThClient(std::move(skt),
23                      cfg, dir_organizer);
24              thclient→start();
25              clients.push_back(thclient);
26          } catch (const SocketException &e) {
27              std::cerr << e.what();
28              return;
29          }
30          this→deleteDeadClients();
31      }
32  }
33
34  void ThAcceptor::deleteDeadClients() {
35      auto it = clients.begin();
36      while (it ≠ clients.end()) {
37          if ((*it)→isAlive()) {
38              ++it;
39          } else {
40              (*it)→join();
41              delete (*it);
42              it = clients.erase(it);
43          }
44      }
45  }
46
47  void ThAcceptor::stop() {
48      int size = clients.size();
49      for (int i = 0; i < size; i++) {
50          clients[i]→stop();
51          clients[i]→join();
52          delete clients[i];
53      }
54      acceptor_skt.close();
55  }
56
57  ThAcceptor::~ThAcceptor() {}
```

```
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #ifndef TP_SYSTCOMMAND_H
6  #define TP_SYSTCOMMAND_H
7
8  #include "server_Command.h"
9  #include <map>
10 #include <string>
11
12 class SystCommand : public Command {
13 private:
14     std::map<std::string, std::string> &cfg;
15     Login& login;
16 public:
17     SystCommand(std::map<std::string, std::string> &cfg, Login& login);
18     std::string execute() override;
19     ~SystCommand() override;
20 };
21
22
23 #endif //TP_SYSTCOMMAND_H
```

```
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #include "server_SystCommand.h"
6
7  #define SYST_KEY "systemInfo"
8  #define UNLOGGED_KEY "clientNotLogged"
9  #define SYST_CODE "215 "
10 #define UNLOGGED_CODE "530 "
11
12 SystCommand::SystCommand(std::map<std::string,
13     std::string> &cfg, Login& login) :
14     cfg(cfg),
15     login(login) {}
16
17 std::string SystCommand::execute() {
18     login.resetIfNotLogged();
19     if (login.userIsLogged()) {
20         return SYST_CODE + cfg.find(SYST_KEY)→second;
21     } else {
22         return UNLOGGED_CODE + cfg.find(UNLOGGED_KEY)→second;
23     }
24 }
25
26 SystCommand::~SystCommand() {}
```

```
1   //
2   // Created by leobellaera on 26/9/19.
3   //
4
5   #ifndef TP_SOCKETEXCEPTION_H
6   #define TP_SOCKETEXCEPTION_H
7
8   #include <stdexcept>
9
10  class SocketException : public std::runtime_error {
11  public:
12      explicit SocketException(const char* error) : runtime_error(error) {}
13  };
14
15  #endif //TP_SOCKETEXCEPTION_H
```

```
1   //
2   // Created by leobellaera on 28/9/19.
3   //
4
5   #ifndef TP_SERVERPROXY_H
6   #define TP_SERVERPROXY_H
7
8   #include "common_Socket.h"
9   #include <string>
10
11  class ServerProxy {
12  private:
13      Socket skt;
14  public:
15      explicit ServerProxy(Socket skt);
16      void receiveClientCommand(std::string& input);
17      void sendMsgToClient(std::string& answer);
18      void stopCommunication();
19      ~ServerProxy();
20  };
21
22  #endif //TP_SERVERPROXY_H
```

```cpp
1  //
2  // Created by leobellaera on 28/9/19.
3  //
4
5  #include "server_ServerProxy.h"
6
7  #define DELIM_CHAR '\n'
8
9  ServerProxy::ServerProxy(Socket skt) :
10     skt(std::move(skt)) {}
11
12 void ServerProxy::receiveClientCommand(std::string& input) {
13     char act;
14     while (true) {
15         skt.recvMessage(&act, 1);
16         if (act ≡ DELIM_CHAR) {
17             return;
18         }
19         input.append(1, act);
20     }
21 }
22
23 void ServerProxy::sendMsgToClient(std::string& answer) {
24     answer.append(1, '\n');
25     skt.sendMessage(answer.c_str(), answer.length());
26 }
27
28 void ServerProxy::stopCommunication() {
29     skt.close();
30 }
31
32 ServerProxy::~ServerProxy() {}
```

```cpp
1  //
2  // Created by leobellaera on 26/9/19.
3  //
4
5  #ifndef TP_SERVERFTP_H
6  #define TP_SERVERFTP_H
7
8  #include "server_DirectoryOrganizer.h"
9  #include "server_ThAcceptor.h"
10 #include "server_CfgMapBuilder.h"
11
12 class ServerFtp {
13 private:
14     DirectoryOrganizer dir_organizer;
15     CfgMapBuilder map_builder;
16     ThAcceptor acceptor_thread;
17 public:
18     ServerFtp(const char* config_path, const char* service, int backlog);
19     void run();
20     ~ServerFtp();
21 };
22
23 #endif //TP_SERVERFTP_H
```

```
1  //
2  // Created by leobellaera on 26/9/19.
3  //
4
5  #include "server_ServerFtp.h"
6  #include <iostream>
7
8  #define STOP_SV_KEY 'q'
9
10 ServerFtp::ServerFtp(const char* config_path,
11         const char* service, int backlog) :
12     map_builder(config_path),
13     acceptor_thread(dir_organizer, map_builder.getMap(), service, backlog) {}
14
15 void ServerFtp::run() {
16     acceptor_thread.start();
17     char c = '\0';
18     while (c ≠ STOP_SV_KEY) {
19         std::cin >> c;
20     }
21     acceptor_thread.stop();
22     acceptor_thread.join();
23 }
24
25 ServerFtp::~ServerFtp() {}
```

```
1  //
2  // Created by leobellaera on 28/9/19.
3  //
4
5  #ifndef TP_RMDCOMMAND_H
6  #define TP_RMDCOMMAND_H
7
8  #include "server_Command.h"
9  #include "server_Login.h"
10 #include "server_DirectoryOrganizer.h"
11 #include <map>
12 #include <string>
13
14 class RmdCommand : public Command {
15 private:
16     std::string dir_name;
17     std::map<std::string, std::string>& cfg;
18     Login& login;
19     DirectoryOrganizer& dir_organizer;
20 public:
21     RmdCommand(std::string dir_name,
22                std::map<std::string, std::string> &cfg,
23                Login& login, DirectoryOrganizer& d);
24     std::string execute() override;
25     ~RmdCommand() override;
26 };
27
28
29 #endif //TP_RMDCOMMAND_H
```

```
1  //
2  // Created by leobellaera on 28/9/19.
3  //
4
5  #include "server_RmdCommand.h"
6
7  #define RMD_SUCCESS_KEY "rmdSuccess"
8  #define RMD_FAIL_KEY "rmdFailed"
9  #define UNLOGGED_KEY "clientNotLogged"
10 #define UNLOGGED_CODE "530 "
11 #define RMD_FAIL_CODE "550 "
12 #define RMD_SUCCESS_CODE "250 "
13 #define DIR_DELIM '"'
14
15 RmdCommand::RmdCommand(std::string dir_name,
16         std::map<std::string, std::string> &cfg,
17         Login& login, DirectoryOrganizer& d) :
18     dir_name(std::move(dir_name)),
19     cfg(cfg),
20     login(login),
21     dir_organizer(d) {}
22
23 std::string RmdCommand::execute() {
24     login.resetIfNotLogged();
25     if (¬login.userIsLogged()) {
26         return UNLOGGED_CODE + cfg.find(UNLOGGED_KEY)→second;
27     } else {
28         if (¬dir_organizer.removeDir(dir_name)) {
29             return RMD_FAIL_CODE + cfg.find(RMD_FAIL_KEY)→second;
30         } else {
31             std::string ans = RMD_SUCCESS_CODE;
32             ans.append(DIR_DELIM + dir_name + DIR_DELIM);
33             ans.append(' ' + cfg.find(RMD_SUCCESS_KEY)→second);
34             return ans;
35         }
36     }
37 }
38
39 RmdCommand::~RmdCommand() {}
```

```
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #ifndef TP_QUITCOMMAND_H
6  #define TP_QUITCOMMAND_H
7
8  #include <string>
9  #include <map>
10 #include "server_Command.h"
11
12 class QuitCommand : public Command {
13 private:
14     std::map<std::string, std::string> &cfg;
15 public:
16     explicit QuitCommand(std::map<std::string,std::string> &cfg);
17     std::string execute() override;
18     ~QuitCommand() override;
19 };
20
21 #endif //TP_QUITCOMMAND_H
```

```
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #include "server_QuitCommand.h"
6
7  #define ANSWER_KEY "quitSuccess"
8  #define QUIT_CODE "221 "
9
10 QuitCommand::QuitCommand(std::map<std::string, std::string> &cfg) :
11    cfg(cfg) {}
12
13 std::string QuitCommand::execute() {
14    return QUIT_CODE + cfg.find(ANSWER_KEY)→second;
15 }
16
17 QuitCommand::~QuitCommand() {}
```

```
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #ifndef TP_PWDCOMMAND_H
6  #define TP_PWDCOMMAND_H
7
8  #include "server_Command.h"
9  #include <map>
10 #include <string>
11
12 class PwdCommand : public Command {
13 private:
14    std::map<std::string, std::string> &cfg;
15    Login& login;
16 public:
17    PwdCommand(std::map<std::string,std::string> &cfg, Login& login);
18    std::string execute() override;
19    ~PwdCommand() override;
20 };
21
22 #endif //TP_PWDCOMMAND_H
```

```cpp
1   //
2   // Created by leobellaera on 27/9/19.
3   //
4
5   #include "server_PwdCommand.h"
6
7   #define PWD_KEY "currentDirectoryMsg"
8   #define UNLOGGED_KEY "clientNotLogged"
9   #define UNLOGGED_CODE "530 "
10  #define PWD_CODE "257 "
11
12  PwdCommand::PwdCommand(std::map<std::string,std::string> &cfg, Login& login) :
13      cfg(cfg),
14      login(login) {}
15
16
17  std::string PwdCommand::execute() {
18      login.resetIfNotLogged();
19      if (login.userIsLogged()) {
20          return PWD_CODE + cfg.find(PWD_KEY)→second;
21      } else {
22          return UNLOGGED_CODE + cfg.find(UNLOGGED_KEY)→second;
23      }
24  }
25
26  PwdCommand::~PwdCommand() {}
```

```cpp
1   //
2   // Created by leobellaera on 27/9/19.
3   //
4
5   #ifndef TP_PASSCOMMAND_H
6   #define TP_PASSCOMMAND_H
7
8   #include <string>
9   #include <map>
10  #include "server_Login.h"
11  #include "server_Command.h"
12
13  class PassCommand : public Command {
14  private:
15      std::map<std::string, std::string> cfg;
16      std::string pass;
17      Login& login;
18  public:
19      PassCommand(std::string pass,
20              std::map<std::string,std::string>& cfg, Login& login);
21      std::string execute() override;
22      ~PassCommand() override;
23  };
24
25  #endif //TP_PASSCOMMAND_H
```

```cpp
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #include "server_PassCommand.h"
6
7  #define LOGIN_SUCCESS_KEY "loginSuccess"
8  #define LOGIN_FAIL_KEY "loginFailed"
9  #define LOGIN_FAIL_CODE "530 "
10 #define LOGIN_SUCCESS_CODE "230 "
11
12 PassCommand::PassCommand(std::string pass,
13         std::map<std::string,std::string>& cfg, Login& login) :
14     cfg(cfg),
15     pass(std::move(pass)),
16     login(login) {}
17
18 std::string PassCommand::execute() {
19     login.enterPassword(pass);
20     if (login.userIsLogged()) {
21         return LOGIN_SUCCESS_CODE + cfg.find(LOGIN_SUCCESS_KEY)→second;
22     } else {
23         return LOGIN_FAIL_CODE + cfg.find(LOGIN_FAIL_KEY)→second;
24     }
25 }
26
27 PassCommand::~PassCommand() {}
```

```cpp
1  //
2  // Created by leobellaera on 28/9/19.
3  //
4
5  #ifndef TP_MKDCOMMAND_H
6  #define TP_MKDCOMMAND_H
7
8  #include "server_Command.h"
9  #include "server_Login.h"
10 #include "server_DirectoryOrganizer.h"
11 #include <map>
12 #include <string>
13
14 class MkdCommand : public Command {
15 private:
16     std::string dir_name;
17     std::map<std::string, std::string>& cfg;
18     Login& login;
19     DirectoryOrganizer& dir_organizer;
20 public:
21     MkdCommand(std::string dir_name,
22                 std::map<std::string, std::string> &cfg,
23                 Login& login, DirectoryOrganizer& d);
24     std::string execute() override;
25     ~MkdCommand() override;
26 };
27
28
29 #endif //TP_MKDCOMMAND_H
```

```cpp
1  //
2  // Created by leobellaera on 28/9/19.
3  //
4
5  #include "server_MkdCommand.h"
6
7  #define MKD_SUCCESS_KEY "mkdSuccess"
8  #define MKD_FAIL_KEY "mkdFailed"
9  #define UNLOGGED_KEY "clientNotLogged"
10 #define UNLOGGED_CODE "530 "
11 #define MKD_SUCCESS_CODE "257 "
12 #define MKD_FAIL_CODE "550 "
13 #define DIR_DELIM '"'
14
15 MkdCommand::MkdCommand(std::string dir_name, std::map<std::string,
16         std::string> &cfg, Login& login, DirectoryOrganizer& d) :
17         dir_name(std::move(dir_name)),
18         cfg(cfg),
19         login(login),
20         dir_organizer(d) {}
21
22 std::string MkdCommand::execute() {
23     login.resetIfNotLogged();
24     if (¬login.userIsLogged()) {
25         return UNLOGGED_CODE + cfg.find(UNLOGGED_KEY)→second;
26     } else {
27         if (¬dir_organizer.makeDir(dir_name)) {
28             return MKD_FAIL_CODE + cfg.find(MKD_FAIL_KEY)→second;
29         } else {
30             std::string ans = MKD_SUCCESS_CODE;
31             ans.append(DIR_DELIM + dir_name + DIR_DELIM);
32             ans.append(' ' + cfg.find(MKD_SUCCESS_KEY)→second);
33             return ans;
34         }
35     }
36 }
37
38 MkdCommand::~MkdCommand() {}
```

```cpp
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #ifndef TP_LOGIN_H
6  #define TP_LOGIN_H
7
8  #include <cstdint>
9  #include <map>
10 #include <string>
11
12 class Login {
13 private:
14     uint8_t stage;
15     std::string user;
16     std::string pass;
17 public:
18     explicit Login(std::map<std::string, std::string> &cfg);
19     bool userIsLogged();
20     void enterUser(std::string& user);
21     void enterPassword(std::string& pass);
22     void resetIfNotLogged();
23     ~Login();
24 };
25
26
27 #endif //TP_LOGIN_H
```

```
1   //
2   // Created by leobellaera on 27/9/19.
3   //
4
5   #include "server_Login.h"
6
7   #define USER_REQUIRED_STAGE 0
8   #define PASS_REQUIRED_STAGE 1
9   #define LOGGED_STAGE 2
10
11  #define USER_KEY "user"
12  #define PASS_KEY "password"
13
14  #define USER_COMMAND "USER"
15  #define PASS_COMMAND "PASS"
16
17  Login::Login(std::map<std::string,std::string> &cfg) :
18      stage(0) {
19      user = cfg.find(USER_KEY)→second;
20      pass = cfg.find(PASS_KEY)→second;
21  }
22
23  bool Login::userIsLogged() {
24      return (stage ≡ LOGGED_STAGE);
25  }
26
27  void Login::resetIfNotLogged() {
28      if (stage ≠ LOGGED_STAGE) {
29          stage = USER_REQUIRED_STAGE;
30      }
31  }
32
33  void Login::enterUser(std::string& user) {
34      if (this→user ≡ user) {
35          stage = PASS_REQUIRED_STAGE;
36      }
37  }
38
39  void Login::enterPassword(std::string& pass) {
40      if (this→pass ≡ pass ∧ stage ≡ PASS_REQUIRED_STAGE) {
41          stage = LOGGED_STAGE;
42      }
43  }
44
45  Login::~Login() {}
```

```
1   //
2   // Created by leobellaera on 28/9/19.
3   //
4
5   #ifndef TP_LISTCOMMAND_H
6   #define TP_LISTCOMMAND_H
7
8   #include "server_Command.h"
9   #include "server_Login.h"
10  #include <map>
11  #include <string>
12  #include "server_DirectoryOrganizer.h"
13
14  class ListCommand : public Command {
15  private:
16      std::map<std::string, std::string> &cfg;
17      Login& login;
18      DirectoryOrganizer& dir_organizer;
19  public:
20      ListCommand(std::map<std::string, std::string> &cfg,
21                  Login& login,
22                  DirectoryOrganizer& d);
23      std::string execute() override;
24      ~ListCommand() override;
25  };
26
27  #endif //TP_LISTCOMMAND_H
```

```cpp
1   //
2   // Created by leobellaera on 28/9/19.
3   //
4
5   #include "server_ListCommand.h"
6
7   #define UNLOGGED_KEY "clientNotLogged"
8   #define LIST_BEGIN_KEY "listBegin"
9   #define LIST_END_KEY "listEnd"
10
11  #define UNLOGGED_CODE "530 "
12  #define LIST_BEGIN_CODE "150 "
13  #define LIST_END_CODE "226 "
14
15  #define DELIM_CHAR '\n'
16
17  ListCommand::ListCommand(std::map<std::string, std::string> &cfg,
18                           Login& login, DirectoryOrganizer& dir_org) :
19      cfg(cfg),
20      login(login),
21      dir_organizer(dir_org) {}
22
23  std::string ListCommand::execute() {
24      login.resetIfNotLogged();
25      std::string answer;
26      if (login.userIsLogged()) {
27          answer.append(LIST_BEGIN_CODE +
28          cfg.find(LIST_BEGIN_KEY)→second + DELIM_CHAR);
29          answer.append(dir_organizer.getDirectories());
30          answer.append(LIST_END_CODE + cfg.find(LIST_END_KEY)→second);
31          return std::move(answer);
32      } else {
33          return UNLOGGED_CODE + cfg.find(UNLOGGED_KEY)→second;
34      }
35  }
36
37  ListCommand::~ListCommand() {}
```

```cpp
1   //
2   // Created by leobellaera on 27/9/19.
3   //
4
5   #ifndef TP_HELPCOMMAND_H
6   #define TP_HELPCOMMAND_H
7
8
9   #include "server_Command.h"
10  #include <string>
11  #include <map>
12
13  class HelpCommand : public Command {
14  private:
15      std::map<std::string, std::string> &cfg;
16      Login& login;
17  public:
18      HelpCommand(std::map<std::string,std::string> &cfg, Login& login);
19      std::string execute() override;
20      ~HelpCommand() override;
21  };
22
23  #endif //TP_HELPCOMMAND_H
```

```
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #include "server_HelpCommand.h"
6
7  #define HELP_KEY "commands"
8  #define UNLOGGED_KEY "clientNotLogged"
9  #define UNLOGGED_CODE "530 "
10 #define HELP_CODE "214 "
11
12 HelpCommand::HelpCommand(std::map<std::string,std::string> &cfg, Login& login) :
13     cfg(cfg),
14     login(login) {}
15
16 std::string HelpCommand::execute() {
17     login.resetIfNotLogged();
18     if (login.userIsLogged()) {
19         return HELP_CODE + cfg.find(HELP_KEY)→second;
20     } else {
21         return UNLOGGED_CODE + cfg.find(UNLOGGED_KEY)→second;
22     }
23 }
24
25 HelpCommand::~HelpCommand() {}
```

```
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #ifndef TP_DIRECTORYORGANIZER_H
6  #define TP_DIRECTORYORGANIZER_H
7
8  #include <set>
9  #include <string>
10 #include <mutex>
11
12 class DirectoryOrganizer {
13 private:
14     std::set<std::string> directories;
15     std::mutex m;
16 public:
17     DirectoryOrganizer();
18     bool makeDir(std::string name);
19     bool removeDir(std::string name);
20     std::string getDirectories();
21 };
22
23
24 #endif //TP_DIRECTORYORGANIZER_H
```

```cpp
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #include "server_DirectoryOrganizer.h"
6  #define DIR_PREFIX "drwxrwxrwx 0 1000 1000 4096 Sep 24 12:34 "
7
8  DirectoryOrganizer::DirectoryOrganizer() {}
9
10 bool DirectoryOrganizer::makeDir(std::string name) {
11     std::unique_lock<std::mutex> lock(m);
12     return directories.emplace(name).second;
13 }
14
15 bool DirectoryOrganizer::removeDir(std::string name) {
16     std::unique_lock<std::mutex> lock(m);
17     if (directories.find(name) ≡ directories.end()) {
18         return false;
19     }
20     directories.erase(name);
21     return true;
22 }
23
24 std::string DirectoryOrganizer::getDirectories() {
25     std::unique_lock<std::mutex> lock(m);
26     std::string ret;
27     for (const auto & dir : directories) {
28         ret.append(DIR_PREFIX + dir);
29         ret.append("\n");
30     }
31     return std::move(ret);
32 }
```

```cpp
1  //
2  // Created by leobellaera on 29/9/19.
3  //
4
5  #include "server_ServerFtp.h"
6  #include "server_CfgMapBuilderException.h"
7  #include <iostream>
8
9  #define INVALID_ARGS_AMOUNT_MSG "Invalid number of arguments"
10 #define UNKNOWN_ERROR_MSG "An unknown error occurred in the execution"
11 #define CONFIG_PATH_POS 2
12 #define PORT_POS 1
13 #define BACKLOG 50
14
15 int main(int argc, char* argv[]) {
16     if (argc ≠ 3) {
17         std::cerr << INVALID_ARGS_AMOUNT_MSG << std::endl;
18         return 1;
19     }
20     try {
21         ServerFtp sv(argv[CONFIG_PATH_POS], argv[PORT_POS], BACKLOG);
22         sv.run();
23     } catch (const std::exception &e) {
24         std::cerr << e.what();
25         return 1;
26     } catch (...) {
27         std::cerr << UNKNOWN_ERROR_MSG << std::endl;
28     }
29     return 0;
30 }
```

```
1   //
2   // Created by leobellaera on 27/9/19.
3   //
4
5   #ifndef TP_COMMAND_H
6   #define TP_COMMAND_H
7
8   #include <map>
9   #include <string>
10  #include "server_Login.h"
11  #include "server_DirectoryOrganizer.h"
12
13  class Command {
14  public:
15      static Command* make_command(std::map<std::string,std::string>& cfg,
16                                   std::string& command,
17                                   Login& login,
18                                   DirectoryOrganizer& dir_org);
19      virtual std::string execute() = 0;
20      virtual ~Command();
21  };
22
23  #endif //TP_COMMAND_H
```

```
1   //
2   // Created by leobellaera on 27/9/19.
3   //
4
5   #include <sstream>
6   #include "server_Command.h"
7   #include "server_UserCommand.h"
8   #include "server_PassCommand.h"
9   #include "server_SystCommand.h"
10  #include "server_QuitCommand.h"
11  #include "server_HelpCommand.h"
12  #include "server_PwdCommand.h"
13  #include "server_UnknownCommand.h"
14  #include "server_MkdCommand.h"
15  #include "server_RmdCommand.h"
16  #include "server_ListCommand.h"
17
18  #define USER_COMMAND "USER"
19  #define PASS_COMMAND "PASS"
20  #define SYST_COMMAND "SYST"
21  #define HELP_COMMAND "HELP"
22  #define PWD_COMMAND "PWD"
23  #define QUIT_COMMAND "QUIT"
24  #define LIST_COMMAND "LIST"
25  #define MKD_COMMAND "MKD"
26  #define RMD_COMMAND "RMD"
27
28  Command* Command::make_command(std::map<std::string, std::string>& cfg,
29          std::string& command, Login& login, DirectoryOrganizer& dir_org) {
30      std::string first_arg = command.substr(0, command.find(' '));
31      std::string second_arg = command.substr(command.find(' ') + 1,
32              command.length());
33      if (first_arg ≡ USER_COMMAND) {
34          return new UserCommand(second_arg, cfg, login);
35
36      } else if (first_arg ≡ PASS_COMMAND) {
37          return new PassCommand(second_arg, cfg, login);
38
39      } else if (first_arg ≡ SYST_COMMAND) {
40          return new SystCommand(cfg, login);
41
42      } else if (first_arg ≡ QUIT_COMMAND) {
43          return new QuitCommand(cfg);
44
45      } else if (first_arg ≡ HELP_COMMAND) {
46          return new HelpCommand(cfg, login);
47
48      } else if (first_arg ≡ PWD_COMMAND) {
49          return new PwdCommand(cfg, login);
50
51      } else if (first_arg ≡ MKD_COMMAND) {
52          return new MkdCommand(second_arg, cfg, login, dir_org);
53
54      } else if (first_arg ≡ RMD_COMMAND){
55          return new RmdCommand(second_arg, cfg, login, dir_org);
56
57      } else if (first_arg ≡ LIST_COMMAND) {
58          return new ListCommand(cfg, login, dir_org);
59
60      } else {
61          return new UnknownCommand(cfg, login);
62      }
63  }
64
65  Command::~Command() {}
```

```cpp
//
// Created by leobellaera on 25/9/19.
//

#ifndef TP_CFGMAPBUILDER_H
#define TP_CFGMAPBUILDER_H

#include <fstream>
#include <map>
#include <string>

class CfgMapBuilder {
private:
    std::ifstream file;
    std::map<std::string,std::string> data;
    void buildMap();
public:
    explicit CfgMapBuilder(const char* file_path);
    std::map<std::string,std::string>& getMap();
    ~CfgMapBuilder();
};

#endif //TP_CFGMAPBUILDER_H
```

```cpp
//
// Created by leobellaera on 29/9/19.
//

#ifndef TP_CFGMAPBUILDEREXCEPTION_H
#define TP_CFGMAPBUILDEREXCEPTION_H

#include <stdexcept>

class CfgMapBuilderException : public std::runtime_error {
public:
    explicit CfgMapBuilderException(const char* error) : runtime_error(error) {}
};

#endif //TP_CFGMAPBUILDEREXCEPTION_H
```

```cpp
1  //
2  // Created by leobellaera on 25/9/19.
3  //
4
5  #include "server_CfgMapBuilder.h"
6  #include "server_CfgMapBuilderException.h"
7  #include <sstream>
8  #define DELIM_CHAR '='
9  #define OPENING_ERROR_MSG "An error occurred while\
10  trying to open the config file.\n"
11
12 #define READING_ERROR_MSG "An error occurred while\
13  trying to read the config file.\n"
14
15 CfgMapBuilder::CfgMapBuilder(const char* file_path) {
16     file.exceptions(std::ifstream::badbit);
17     file.open(file_path);
18     if (¬file.is_open()) {
19         throw CfgMapBuilderException(OPENING_ERROR_MSG);
20     }
21     try {
22         this→buildMap();
23     } catch (std::ios_base::failure& e) {
24         throw CfgMapBuilderException(READING_ERROR_MSG);
25     }
26 }
27
28 void CfgMapBuilder::buildMap() {
29     while (¬file.eof()) {
30         std::string aux;
31         std::string key;
32         std::string value;
33
34         std::getline(file, aux);
35         std::istringstream line_stream(aux);
36
37         std::getline(line_stream, key, DELIM_CHAR);
38         std::getline(line_stream, value, DELIM_CHAR);
39         data.emplace(key, value);
40     }
41 }
42
43 std::map<std::string,std::string>& CfgMapBuilder::getMap(){
44     return data;
45 }
46
47 CfgMapBuilder::~CfgMapBuilder() {}
```

```cpp
1  //
2  // Created by leobellaera on 26/9/19.
3  //
4
5  #ifndef TP_ACCEPTORSOCKET_H
6  #define TP_ACCEPTORSOCKET_H
7
8  #include "common_Socket.h"
9
10 class AcceptorSocket {
11 private:
12     int fd;
13     int getAddressInfo(struct addrinfo **addrinfo_ptr, const char* service);
14     int bindAndListen(int backlog, const char* service);
15     int listen(int backlog);
16     int bind(struct addrinfo* ptr);
17 public:
18     AcceptorSocket(int backlog, const char* service);
19     Socket accept();
20     void close();
21     ~AcceptorSocket();
22 };
23
24
25 #endif //TP_ACCEPTORSOCKET_H
```

```
1   //
2   // Created by leobellaera on 26/9/19.
3   //
4
5   #include "server_AcceptorSocket.h"
6   #include "common_SocketException.h"
7   #include <cerrno>
8   #include <sys/socket.h>
9   #include <netdb.h>
10  #include <unistd.h>
11  #include <cstring>
12  #include <iostream>
13
14  #define ACCEPT_ERROR_MSG "Error occurred while trying to accept client\n"
15  #define EMPTY_MSG ""
16
17  AcceptorSocket::AcceptorSocket(int backlog, const char* service) :
18      fd(-1) {
19      if (this→bindAndListen(backlog, service) ≡ 1) {
20          this→close();
21          throw SocketException(EMPTY_MSG);
22      }
23  }
24
25  Socket AcceptorSocket::accept() {
26      int peer_fd = ::accept(fd, nullptr, nullptr);
27      if (peer_fd ≡ -1){
28          throw SocketException(ACCEPT_ERROR_MSG);
29      }
30      return std::move(Socket(peer_fd));
31  }
32
33  int AcceptorSocket::bindAndListen(int backlog, const char* service) {
34      struct addrinfo *ptr = nullptr;
35      int s = this→getAddressInfo(&ptr, service);
36      if (s ≠ 0) {
37          std::cerr << "Error in getaddrinfo: " << gai_strerror(s) << std::endl;
38          return 1;
39      }
40      fd = socket(ptr→ai_family, ptr→ai_socktype, ptr→ai_protocol);
41      if (fd ≡ -1) {
42          std::cerr << "Error: " << strerror(errno) << std::endl;
43          freeaddrinfo(ptr);
44          return 1;
45      }
46      if (bind(ptr) ≡ 1) {
47          freeaddrinfo(ptr);
48          return 1;
49      }
50      freeaddrinfo(ptr);
51      if (listen(backlog) ≡ 1) {
52          return 1;
53      }
54      return 0;
55  }
56
57  int AcceptorSocket::bind(struct addrinfo* ptr) {
58      int s = ::bind(fd, ptr→ai_addr, ptr→ai_addrlen);
59      if (s ≡ -1) {
60          std::cerr << "Error: " << strerror(errno) << std::endl;
61          return 1;
62      }
63      return 0;
64  }
65
66  int AcceptorSocket::listen(int backlog) {
```

```
67      int s = ::listen(fd, backlog);
68      if (s ≡ -1) {
69          std::cerr << "Error: " << strerror(errno) << std::endl;
70          return 1;
71      }
72      return 0;
73  }
74
75  int AcceptorSocket::getAddressInfo(struct addrinfo **addrinfo_ptr,
76          const char* service) {
77      struct addrinfo hints;
78      memset(&hints, 0, sizeof(struct addrinfo));
79      hints.ai_family = AF_INET;
80      hints.ai_socktype = SOCK_STREAM;
81      hints.ai_flags = AI_PASSIVE;
82      return getaddrinfo(nullptr, service, &hints, addrinfo_ptr);
83  }
84
85  void AcceptorSocket::close() {
86      if (fd ≠ -1) {
87          shutdown(fd, SHUT_RDWR);
88          ::close(fd);
89      }
90  }
91
92  AcceptorSocket::~AcceptorSocket() {
93      this→close();
94  }
```

```
1  //
2  // Created by leobellaera on 26/9/19.
3  //
4
5  #ifndef TP_SOCKET_H
6  #define TP_SOCKET_H
7  #include <netdb.h>
8
9  class Socket {
10 private:
11     int fd;
12     int connect(const char* host, const char* service);
13     bool establishConnection(addrinfo* result);
14     int getAddressInfo(struct addrinfo **addrinfo_ptr,
15             const char* host, const char* service);
16 public:
17     Socket(const char* host, const char* service);
18     explicit Socket(int fd);
19     Socket(Socket ∧other) noexcept;
20     void sendMessage(const char* buffer, int size);
21     void recvMessage(char* buffer, int size);
22     void close();
23     ~Socket();
24 };
25
26 #endif //TP_SOCKET_H
```

```
1  //
2  // Created by leobellaera on 26/9/19.
3  //
4
5  #ifndef TP_SOCKETEXCEPTION_H
6  #define TP_SOCKETEXCEPTION_H
7
8  #include <stdexcept>
9
10 class SocketException : public std::runtime_error {
11 public:
12     explicit SocketException(const char* error) : runtime_error(error) {}
13 };
14
15 #endif //TP_SOCKETEXCEPTION_H
```

```
1   //
2   // Created by leobellaera on 26/9/19.
3   //
4
5   #include "common_Socket.h"
6   #include "common_SocketException.h"
7   #include <sys/socket.h>
8   #include <iostream>
9   #include <cstring>
10  #include <unistd.h>
11  #include <cstdlib>
12
13  #define SEND_ERROR_MSG "Error occurred while trying to send message\n"
14  #define RECV_ERROR_MSG "Error occurred while trying to receive message\n"
15  #define EMPTY_MSG ""
16
17  Socket::Socket(const char* host, const char* service) :
18      fd(-1) {
19      if (this→connect(host, service) ≡ 1) {
20          this→close();
21          throw SocketException(EMPTY_MSG);
22      }
23  }
24
25  Socket::Socket(int fd) :
26      fd(fd) {}
27
28  Socket::Socket(Socket ∧other) noexcept {
29      this→fd = other.fd;
30      other.fd = -1;
31  }
32
33  int Socket::connect(const char* host, const char* service) {
34      struct addrinfo *result = nullptr;
35      int s = this→getAddressInfo(&result, host, service);
36      if (s ≠ 0) {
37          std::cerr << "Error in getaddrinfo: " << gai_strerror(s) << std::endl;;
38          freeaddrinfo(result);
39          return 1;
40      }
41      if (¬this→establishConnection(result)) {
42          freeaddrinfo(result);
43          return 1;
44      }
45      freeaddrinfo(result);
46      return 0;
47  }
48
49  bool Socket::establishConnection(addrinfo* result) {
50      struct addrinfo* ptr;
51      int s;
52      bool connection_established = false;
53      for (ptr = result; ptr ≠ nullptr ∧ ¬connection_established ;
54              ptr = ptr→ai_next) {
55          fd = socket(ptr→ai_family, ptr→ai_socktype, ptr→ai_protocol);
56          if (fd ≡ -1) {
57              std::cerr << "Error: " << strerror(errno) << std::endl;
58          } else {
59              s = ::connect(fd, ptr→ai_addr, ptr→ai_addrlen);
60              if (s ≡ -1) {
61                  std::cerr << "Error: " << strerror(errno) << std::endl;
62              }
63              connection_established = (s ≠ -1);
64          }
65      }
66      return connection_established;
```

```
67  }
68
69
70  int Socket::getAddressInfo(struct addrinfo **addrinfo_ptr,
71          const char* host, const char* service) {
72      struct addrinfo hints;
73      memset(&hints, 0, sizeof(struct addrinfo));
74      hints.ai_family = AF_INET;
75      hints.ai_socktype = SOCK_STREAM;
76      hints.ai_flags = 0;
77      return getaddrinfo(host, service, &hints, addrinfo_ptr);
78  }
79
80  void Socket::sendMessage(const char* buffer, int size) {
81      int sent = 0;
82      int s = 0;
83      while (sent < size) {
84          s = send(fd, &buffer[sent], size - sent, MSG_NOSIGNAL);
85          if (s ≡ 0 ∨ s ≡ -1) {
86              throw SocketException(SEND_ERROR_MSG);
87          } else {
88              sent += s;
89          }
90      }
91  }
92
93  void Socket::recvMessage(char* buffer, int size) {
94      int received = 0;
95      int s = 0;
96      while (received < size) {
97          s = recv(fd, &buffer[received], size-received, 0);
98          if (s ≡ 0 ∨ s ≡ -1) {
99              throw SocketException(RECV_ERROR_MSG);
100         } else {
101             received += s;
102         }
103     }
104 }
105
106 void Socket::close() {
107     if (fd ≠ -1) {
108         shutdown(fd, SHUT_RDWR);
109         ::close(fd);
110         fd = -1;
111     }
112 }
113
114 Socket::~Socket() {
115     this→close();
116 }
```

```cpp
1  //
2  // Created by leobellaera on 29/9/19.
3  //
4
5  #include <iostream>
6  #include "client_ClientFtp.h"
7  #include "common_SocketException.h"
8
9  #define INVALID_ARGS_AMOUNT_MSG "Invalid number of arguments\n"
10 #define UNKNOWN_ERROR_MSG "An unknown error occurred in the execution\n"
11 #define HOST_POS 1
12 #define SERVICE_POS 2
13
14 int main(int argc, char* argv[]) {
15     if (argc ≠ 3) {
16         std::cerr<<INVALID_ARGS_AMOUNT_MSG;
17         return 1;
18     }
19     try {
20         ClientFtp client(argv[HOST_POS], argv[SERVICE_POS]);
21         client.run();
22     } catch (const SocketException& e) {
23         std::cerr << e.what();
24         return 1;
25     } catch (...) {
26         std::cerr << UNKNOWN_ERROR_MSG;
27         return 1;
28     }
29     return 0;
30 }
```

```cpp
1  //
2  // Created by leobellaera on 26/9/19.
3  //
4
5  #ifndef TP_CLIENTPROXY_H
6  #define TP_CLIENTPROXY_H
7
8  #include "common_Socket.h"
9  #include <string>
10
11 class ClientProxy {
12 private:
13     Socket skt;
14     void recvSvAnswer(std::string& answer);
15 public:
16     ClientProxy(const char* host, const char* service);
17     bool executeCommand(std::string &command, std::string &answer);
18     bool recvSvMessage(std::string& line);
19     ~ClientProxy();
20 };
21
22 #endif //TP_CLIENTPROXY_H
```

```
1  //
2  // Created by leobellaera on 26/9/19.
3  //
4
5  #include "client_ClientProxy.h"
6  #include "common_SocketException.h"
7  #include <vector>
8  #include <algorithm>
9  #include <iostream>
10
11 #define MSG_DELIM '\n'
12 #define HELP "214"
13 #define SYST "215"
14 #define QUIT "221"
15 #define LIST "226"
16 #define PASS_S "230"
17 #define RMD_S "250"
18 #define PWD "257"
19 #define USER "331"
20 #define COMMAND_ERR "530"
21 #define DIR_FAIL "550"
22
23 ClientProxy::ClientProxy(const char* host, const char* service) :
24     skt(host, service) {}
25
26 bool ClientProxy::executeCommand(std::string &command, std::string &answer) {
27     command.append(1, MSG_DELIM);
28     try {
29         skt.sendMessage(command.c_str(), command.length());
30         this→recvSvAnswer(answer);
31         command.pop_back();
32     } catch (const SocketException &e) {
33         std::cerr << e.what() << std::endl;
34         return false;
35     }
36     return true;
37 }
38
39 void ClientProxy::recvSvAnswer(std::string& answer) {
40     answer.clear();
41     bool ans_totally_received = false;
42     std::string line;
43     while (¬ans_totally_received) {
44         line.clear();
45         ans_totally_received = this→recvSvMessage(line);
46         answer.append(line + MSG_DELIM);
47     }
48 }
49
50 bool ClientProxy::recvSvMessage(std::string &line) {
51     const std::vector<std::string> tokens = {HELP, SYST, QUIT,
52         LIST, PASS_S, RMD_S, PWD, USER, COMMAND_ERR, DIR_FAIL};
53     bool ans_totally_received = false;
54     char received_char = '\0';
55     while (true) {
56         skt.recvMessage(&received_char, 1);
57         if (received_char ≡ MSG_DELIM) {
58             break;
59         }
60         line.append(1, received_char);
61     }
62     std::string code = line.substr(0, 3);
63     if (std::find(tokens.begin(), tokens.end(), code) ≠ tokens.end()) {
64         ans_totally_received = true;
65     }
66     return ans_totally_received;
```

```
67 }
68
69 ClientProxy::~ClientProxy() {}
```

```
1  //
2  // Created by leobellaera on 26/9/19.
3  //
4
5  #ifndef TP_CLIENTFTP_H
6  #define TP_CLIENTFTP_H
7
8  #include "common_Socket.h"
9  #include "client_ClientProxy.h"
10
11 class ClientFtp {
12 private:
13     ClientProxy proxy;
14 public:
15     ClientFtp(const char* host, const char* service);
16     void run();
17     ~ClientFtp();
18 };
19
20 #endif //TP_CLIENTFTP_H
```

```
1  //
2  // Created by leobellaera on 26/9/19.
3  //
4
5  #include "client_ClientFtp.h"
6  #include <iostream>
7  #include <string>
8
9  #define QUIT_COMMAND "QUIT"
10
11 ClientFtp::ClientFtp(const char* host, const char* service) :
12     proxy(host, service) {}
13
14 void ClientFtp::run() {
15     std::string command;
16     std::string sv_msg;
17     proxy.recvSvMessage(sv_msg); //server welcome msg
18     std::cout<<sv_msg<<std::endl;
19
20     while (true) {
21         std::getline(std::cin, command);
22         if (std::cin.eof()) break;
23         if (¬proxy.executeCommand(command, sv_msg)) {
24             return;
25         }
26         std::cout << sv_msg;
27         if (command ≡ QUIT_COMMAND) {
28             return;
29         }
30     }
31 }
32
33 ClientFtp::~ClientFtp() {}
```