Contents

1 common_message.h 2

2 common_message.cpp 3

3 common_define.h 4

4 common_thread.h 6

5 common_socket.h 7

6 common_socket.cpp 9

7 common_proxy.h 13

8 common_proxy.cpp 15

9 client_main.cpp 17

10 client_proxy_server.h 18

11 client_proxy_server.cpp 20

12 client_column_record.h 22

13 client_column_record_wk.h 23

14 client_parser_protocolo_workspaces.h 24

15 client_parser_protocolo_workspaces.cpp 25

16 client_parser_protocolo_morph.h 26

17 client_parser_protocolo_morph.cpp 27

18 client_morph_window.h 29

19 client_morph_window.cpp 32

20 client_select_wk_window.h 39

21 client_select_wk_window.cpp 41

22 client_morph.h 45

23 client_morph.cpp 48

24 server_main.cpp 50

25 server_workspace.h **51**

26 server_workspace.cpp **52**

27 server_parser.h **53**

28 server_parser.cpp **56**

29 server_accepter.h **69**

30 server_accepter.cpp **71**

31 server_server.h **72**

32 server_server.cpp **75**

33 server_proxy_client.h **79**

34 server_proxy_client.cpp **82**

35 server_object.h **88**

36 server_object.cpp **94**

37 server_virtual_machine.h **107**

38 server_virtual_machine.cpp **108**

39 server_mode_selector.h **110**

40 server_mode_selector.cpp **111**

41 server_parser_protocolo_servidor.h **112**

42 server_parser_protocolo_servidor.cpp **113**

1 common_message.

```
#ifndef COMMON_TYPES_H_
#define COMMON_TYPES_H_

#include <string>
#include <netinet/in.h>
#include <cstring>

class Message {
private:
    size_t length;
    char instr;
    std::string message;

public:
    Message(size_t length, char command, std::string message);
    Message();

    /** Este metodo devuelve el objeto convertido a std::string.
     *
     */
    std::string toString();

    /** Devuelve la longitud del mensaje
     *
     */
    size_t getLength() const;

    /** Fija la longitud del mensaje
     * @param len nueva longitud
     */
    void setLength(const size_t len);

    /** Devuelve el texto del mensaje-
     *
     */
    std::string getMessage() const;

    /** Fija el text del mensaje.
     * @param str nuevo mensaje
     */
    void setMessage(const char* str);

    /** Fija el comando del mensaje.
     * @param cmd nuevo comando.
     */
    void setCommand(const char cmd);

    /** Devuelve el comando del mensaje.
     *
     */
    char getCommand() const;
};

#endif /* COMMON_TYPES_H_ */
```

2 common_message.cpp

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

Message::Message(size_t length, char command, std::string message) :
        length(length), instr(command), message(message) {
}
Message::Message() {
}

std::string Message::toString() {
    std::string command;
    char *strLength = new char[sizeof(int) + 1];
    size_t length = htons(this->length);
    memcpy(strLength, &length, sizeof(int));

    command += std::string(strLength);
    command += this->instr;
    command += message;

    delete[] strLength;
    return command;
}

size_t Message::getLength() const {
    return length;
}

void Message::setLength(const size_t len) {
    this->length = len;
}

std::string Message::getMessage() const {
    return message;
}

void Message::setMessage(const char* str) {
    message = std::string(str);
}

void Message::setCommand(const char cmd) {
    this->instr = cmd;
}

char Message::getCommand() const {
    return instr;
}
```

3 common_define.h

```
#ifndef _COMMON_DEFINE_H_
#define _COMMON_DEFINE_H_

// Mensajes que envia el servidor y recibe el cliente
#define ERRORMESSAGE 0x50
#define OK_MSG_MORPH 0x00
#define OK_MSG_SELECT_WKS 0x01

// Mensajes que envia el cliente y recibe el servidor
//Los comandos 0x0<x> se ejecutan en el contexto lobby
#define EXEC_LOBBY_CMD 0x01
#define SHOW_LOBBY 0x02

//Los comandos 0x[1]<x> se ejecutan en el contexto local del cliente
#define EXEC_LOCAL_CMD 0x10
#define EXEC_REFRESH 0x11
#define SET_OBJ_NAME 0x12
#define SET_CODESEGMENT 0x13
#define ADD_SLOT 0x14
#define REMOVE_SLOT 0x15
#define SWAP_MUTABILITY 0x16
#define GET_SLOT_OBJ 0x17
#define GO_BACK 0x18

//Los comandos 0x[2]<x> comandos relacionados a solicitudes de Workspaces
#define AVAILABLE_WKS 0x20
#define LOAD_WK 0x21
#define NEW_WK 0x22
#define DELETE_WK 0x23
#define CLOSE_WK 0x24

//Protocolo
const std::string FALSE_BIN = "0";
const std::string TRUE_BIN = "1";
#define CHAR_SEPARADOR '~'

//Operadores
const std::string OP_SUMA = "+";
const std::string OP_RESTA = "-";
const std::string OP_MULTIPLICACION = "*";
const std::string OP_DIVISION = "/";
const std::string OP_DISTINTO = "!=";
const std::string OP_IGUAL = "==";

//Parser
const std::string NIL = "nil";
const std::string TRUE_STR = "true";
const std::string FALSE_STR = "false";
const std::string OP_ASIGNACION = ":";
const std::string OP_SLOT_INMUTABLE = "=";
const std::string OP_SLOT_MUTABLE = "<-";
const std::string PUNTO = ".";
const std::string OP_ARG = ":";
const std::string SLOT_LIST_SEP = "|";
const std::string OP_PARENT = "*";
```

```
const std::string OP_NATIVE_METHOD = "(*)";
const std::string P_LEFT = "(";
const std::string P_RIGHT = ")";

//Nombres objetos default
const std::string NIL_OBJ = "nil";
const std::string BOOLEAN_OBJ = "boolean";
const std::string STRING_OBJ = "string";
const std::string NUMBER_OBJ = "number";
const std::string NATIVE_METHOD = "native";
const std::string COMPLEX_OBJ = "object";

//Otras Constantes
const std::string LOBBY = "lobby";
#define ID_LOBBY 0
const std::string SELF = "self";
const std::string COMPLEX_PREVIEW = "...";

//Metodos nativos
const std::string PRINT_METHOD = "print";
const std::string PRINTOBJ_METHOD = "printObj";
const std::string CLONE_METHOD = "clone";
const std::string ADD_SLOTS_METHOD = "_AddSlots";
const std::string REMOVE_SLOTS_METHOD = "_RemoveSlots";
const std::string COLLECT_METHOD = "collect";

//Nombre del archivo glade
const std::string GLADE_FILE = "./windows.glade";

//Server
#define SERVER_QUIT_CHAR 'q'

#endif /* _COMMON_DEFINE_H_ */
```

4 common_thread.h

```
#ifndef _THREAD_H_
#define _THREAD_H_
#include <thread>
#include <utility>

/** Encapsula los metodos para iniciar, correr y joinear hilos.
 *
 */
class Thread {
private:
    std::thread thread;

public:
    Thread() {
    }

    void start() {
        thread = std::thread(&Thread::run, this);
    }

    void join() {
        thread.join();
    }

    virtual void run() = 0;
    virtual ~Thread() {
    }

private:
    // deshabilito el constructor por copia
    Thread(const Thread&) = delete;
    // deshabilito el operador de asignacion para evitar que se
    // copie el objeto.
    Thread& operator=(const Thread&) = delete;

    // constructor por movimiento.
    //
    Thread(Thread&& other) {
        this->thread = std::move(other.thread);
    }

    Thread& operator=(Thread&& other) {
        this->thread = std::move(other.thread);
        return *this;
    }
};

#endif /* _THREAD_H_ */
```

5 common_socket.h

```
#ifndef COMMON_SOCKET_H_
#define COMMON_SOCKET_H_

#ifndef _POSIX_C_SOURCE
#define _POSIX_C_SOURCE 200112L
#endif

#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <string.h>
#include <unistd.h>
#include <string>
#include <cstring>
#include <stdexcept>

/** Representa una encapsulacion de los sockets provistos
 * por el sistema operativo que estan en las librerias de Unix.
 *
 */
class Socket {
private:
    std::string hostname;
    uint32_t port;
    int socket_fd;
    int accepted_socket_fd;
    struct addrinfo hints;
    struct addrinfo addr;
    struct addrinfo *ptr;
    void initialize(uint32_t flags);
    bool _shutdown;

public:
    /** Constructor
     * @param hostname IP del servidor
     * @param port puerto a escuchar
     */
    Socket(std::string hostname, uint32_t port);

    /** Constructor
     * @param port puerto a escuchar
     */
    explicit Socket(uint32_t port);

    /// Constructor por copia
    Socket(const Socket&);

    /// Destructor
    ~Socket();

    /// Metodo que sirve para escuchar en un determinado puerto
    void bind_and_listen();

    /// Metodo que se usa para conectar un cliente a un servidor.
    void connect();
```

```
    /* Metodo que sirve para aceptar un nuevo cliente. Devuelve un puntero
     * a Socket.
     */
    Socket* accept();

    /** Metodo que sirve para enviar un mensaje
     * @param buffer desde donde leer los datos a enviar
     * @param length tamanio en bytes de los datos a enviar
     */
    void send(const char *buffer, uint32_t length);

    /** Metodo que sirve para recibir un mensaje
     * @param buffer donde guardar los datos recibidos
     * @param length tamanio en bytes recibidos
     */
    int receive(char *buffer, uint32_t length);

    /// Cierra el socket y libera los recursos.
    void shutdown();

    /// Operador de asignacion deshabilitado
    Socket& operator=(const Socket&) = delete;
};

#endif /* COMMON_SOCKET_H_ */
```

6 common_socket.cpp

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

Socket::Socket(std::string hostname, uint32_t port) {
    this->socket_fd = 0;
    this->ptr = nullptr;
    this->port = port;
    this->hostname = hostname;
    this->_shutdown = false;
    this->accepted_socket_fd = 0;
    initialize(0);
}

Socket::Socket(uint32_t port) {
    this->socket_fd = 0;
    this->ptr = nullptr;
    this->port = port;
    this->_shutdown = false;
    this->accepted_socket_fd = 0;
    initialize(AI_PASSIVE);
}

Socket::Socket(const Socket &sck) {
    this->ptr = nullptr;
    this->accepted_socket_fd = 0;
    this->_shutdown = false;
    this->port = sck.port;
    this->socket_fd = sck.accepted_socket_fd;
}

Socket::~Socket() {
    shutdown();
}

void Socket::initialize(uint32_t flags) {
    std::memset(&hints, 0, sizeof(struct addrinfo));
    std::memset(&addr, 0, sizeof(struct addrinfo));

    hints.ai_family = AF_INET; /* IPv4 (or AF_INET6 for IPv6)     */
    hints.ai_socktype = SOCK_STREAM; /* TCP  (or SOCK_DGRAM for UDP)    */
    hints.ai_flags = flags;

    std::string __port = std::to_string(port);
    int s = ::getaddrinfo(NULL, __port.c_str(), &hints, &ptr);
    if (s < 0)
        throw std::runtime_error("Error␣en␣la␣llamada␣a␣getaddrinfo.");

    this->socket_fd = ::socket(ptr->ai_family, ptr->ai_socktype,
            ptr->ai_protocol);

    if (this->socket_fd < 0) {
        freeaddrinfo(ptr);
        throw std::runtime_error("No␣se␣pudo␣crear␣el␣socket.");
    }
```

```
    if (!flags) {
        int val = 1;
        s = setsockopt(this->socket_fd, SOL_SOCKET, SO_REUSEADDR, &val,
                sizeof(val));

        if (s < 0) {
            freeaddrinfo(ptr);
            throw std::runtime_error("Error en la llamada a setsockopt");
        }
    }
}

Socket* Socket::accept() {
    accepted_socket_fd = ::accept(this->socket_fd, NULL, NULL);
    if (accepted_socket_fd < 0) {
        std::string error("Error en la llamada a accept().\nError: ");
        std::string error_description(::strerror(errno));
        error += error_description;
        throw std::runtime_error(error);
    }

    Socket *sck = new Socket(*this);
    return sck;
}

int Socket::receive(char *buffer, uint32_t length) {
    uint32_t received = 0;
    int s = 0;
    bool valid = true;
    while (received < length && valid) {
        s = ::recv(this->socket_fd, &buffer[received], length - received,
        MSG_NOSIGNAL);
        if (s < 0) {
            std::string error("Error en la llamada a recv().\nError: ");
            std::string error_description(::strerror(errno));
            error += error_description;
            throw std::runtime_error(error);
        } else if (s == 0) {
            valid = false;
        } else {
            received += s;
        }
    }
    return s;
}

void Socket::connect() {
    struct addrinfo *result;

    bool connected = false;
    this->port = port;

    for (result = ptr; result != NULL && connected == false;
            result = result->ai_next) {
        if (this->socket_fd == -1) {
        } else {
            int s = ::connect(this->socket_fd, result->ai_addr,
```

```
                    result->ai_addrlen);
            if (s == -1) {
                close(this->socket_fd);
                throw std::runtime_error("Error␣en␣la␣llamada␣a␣connect().");
            }
            connected = (s != -1);  // nos conectamos?
        }
    }
// libero la lista de direcciones
    freeaddrinfo(result);
}

void Socket::shutdown() {
    if (ptr) {
        freeaddrinfo(ptr);
        ptr = nullptr;
    }
    if (!_shutdown) {
        ::shutdown(this->socket_fd, SHUT_RDWR);
        ::close(this->socket_fd);
        _shutdown = true;
    }
}

void Socket::send(const char *buffer, uint32_t length) {
    int s = 0;
    uint32_t sent = 0;
    int valid = true;
    while (sent < length && valid) {
        s = ::send(this->socket_fd, &buffer[sent], length - sent, MSG_NOSIGNAL);
        if (s < 0) {  // error inesperado
            throw std::runtime_error("Error␣en␣la␣llamada␣a␣send().");
        } else if (s == 0) {
            valid = false;
        } else {
            sent += s;
        }
    }
}

void Socket::bind_and_listen() {
    int s = ::bind(this->socket_fd, ptr->ai_addr, ptr->ai_addrlen);

    if (s < 0) {
        std::string error("Error␣en␣la␣llamada␣a␣bind().\nError:␣");
        std::string error_description(::strerror(errno));
        error += error_description;
        throw std::runtime_error(error);
    }

    // Puedo mantener en espera 10 clientes antes de aceptarlos
    s = ::listen(this->socket_fd, 10);

    if (s < 0) {
        std::string error("Error␣en␣la␣llamada␣a␣listen().\nError:␣");
        std::string error_description(::strerror(errno));
        error += error_description;
```

```
        throw std::runtime_error(error);
    }
}
```

7 common_proxy.h

```
#ifndef COMMON_PROXY_H_
#define COMMON_PROXY_H_

#include "common_socket.h"
#include "common_thread.h"
#include "common_message.h"
#include "common_define.h"
#include "server_server.h"
#include "server_object.h"
#include <string>
#include <vector>

/** Esta clase es la base para las clases ProxyClient y ProxyServer.
 * Contiene los metodos y atributos comunes a ambos.
 *
 */
class Proxy: public Thread {
public:
    /** Constructor.
     * @param socket socket sobre el cual trabajar
     */
    Proxy(Socket &socket);

    /** Constructor por copia eliminado
     *
     */
    Proxy(const Proxy&) = delete;

    /** Constructor por moviemiento eliminado
     *
     */
    Proxy(Proxy&&) = delete;

    /** Operador asignacion eliminado
     *
     */
    Proxy& operator=(const Proxy&) = delete;

    /** Operador asignacion por moviemiento eliminado
     *
     */
    Proxy& operator=(Proxy&&) = delete;

    /** Indica si la conexion finalizo.
     *
     */
    bool is_finished();

    /** Interrumpe la ejecucion del proxy.
     *
     */
    void interrupt();

protected:
    /// Socket interno
```

```
    Socket &serverSocket;

    /// Flag que indica si hay que interrumpir la recepcion de mensajes.
    bool _interrupt;

    /// Flag que indica si finalizo
    bool finished;

    /// Mensaje recibido
    Message message;

    /** Envia un mensaje
     * @message del tipo command_t con el mensaje
     */
    void send(Message &message);

    /** Recibe datos. Una vez que recibe los datos los guarda
     * internamente.
     *
     */
    virtual int receive();

    /** Envia un mensaje de error
     * @param msg std::string con el mensaje a enviar con la descripcion
     * del error
     */
    void sendError(std::string msg);

    /** Envia un mensaje de OK para el Morph
     * @param msg con el resultado de la operacion.
     */
    void sendOK(std::string msg);

    /** Envia un mensaje de OK para la ventana selectora de Workspaces
     * @param msg con el resultado de la operacion.
     */
    void sendOKWks(std::string msg);
};

#endif /* COMMON_PROXY_H_ */
```

8 common_proxy.cpp

```
#include "common_proxy.h"

Proxy::Proxy(Socket &socket) :
        serverSocket(socket) {
    finished = false;
    _interrupt = false;
}

int Proxy::receive() {
    char *strLen = new char[sizeof(int)];
    memset(strLen, 0, sizeof(int));

    int s = serverSocket.receive(strLen, sizeof(int));
    if (s == 0) {
        delete[] strLen;
        return s;
    }

    unsigned int len = 0;
    memcpy(&len, strLen, sizeof(int));
    len = ntohl(len);

    char command;
    s = serverSocket.receive(&command, sizeof(char));

    char *strMessage = new char[len + 1];
    serverSocket.receive(strMessage, len);
    strMessage[len] = '\0';

    message.setLength(len);
    message.setCommand(command);
    message.setMessage(strMessage);

    delete[] strMessage;
    delete[] strLen;
    return s;
}

void Proxy::send(Message &response) {
    char command = response.getCommand();

    unsigned int len = response.getLength();
    len = htonl(len);

    char *strLen = new char[sizeof(int)];
    memcpy(strLen, &len, sizeof(int));

    this->serverSocket.send(strLen, sizeof(int));
    this->serverSocket.send(&command, sizeof(char));
    this->serverSocket.send(response.getMessage().c_str(),
            response.getMessage().size());
    delete[] strLen;
}

void Proxy::sendError(std::string msg) {
```

```
    Message response(msg.size(), ERRORMESSAGE, msg);
    send(response);
}

void Proxy::sendOK(std::string msg) {
    Message response(msg.size(), OK_MSG_MORPH, msg);
    send(response);
}

void Proxy::sendOKWks(std::string msg) {
    Message response(msg.size(), OK_MSG_SELECT_WKS, msg);
    send(response);
}

bool Proxy::is_finished() {
    return this->finished;
}

void Proxy::interrupt() {
    this->_interrupt = true;
    serverSocket.shutdown();
}
```

9 client_main.cpp

```
#include <iostream>
#include <fstream>
#include <unistd.h>
#include <vector>
#include <string>
#include <mutex>
#include "client_proxy_server.h"

#define CLIENT_PARAMS 3
#define RET_OK 0
#define RET_NOK 1

#include "client_select_wk_window.h"

int main(int argc, char **argv) {
    try {
        if (argc != CLIENT_PARAMS) {
            std::cout << "Argumentos:␣>>>␣.\\client␣<serverIP>␣<puerto>."
                    << std::endl;
            return RET_NOK;
        }

        std::mutex m;
        std::string server = argv[1];
        uint32_t port = 0;
        try {
            port = std::stoi(*(argv + 2));
        } catch (...) {
            throw std::runtime_error("puerto␣invalido.");
        }

        Morph morph;
        std::vector<std::string> workspaces;
        Socket socket(server, port);
        ProxyServer proxyServer(socket, morph, workspaces, m);

        //Aca se abrio un hilo nuevo que es del proxy
        proxyServer.start();

        auto app = Gtk::Application::create();
        SelectWkWindow window(morph, workspaces, proxyServer, m);
        app->run(*window.getWindow());

        proxyServer.interrupt();
        proxyServer.join();
    } catch (const std::runtime_error &e) {
        std::cout << "Error.␣" << std::endl << e.what() << std::endl;
    } catch (...) {
        std::cout << "Error␣desconocido." << std::endl << std::endl;
    }
    return RET_OK;
}
```

10 client_proxy_server.h

```
#ifndef CLIENT_PROXYSERVER_H_
#define CLIENT_PROXYSERVER_H_

#include "common_socket.h"
#include "common_proxy.h"
#include "client_morph.h"
#include "client_parser_protocolo_morph.h"
#include "client_parser_protocolo_workspaces.h"
#include "common_define.h"
#include <vector>
#include <string>
#include <mutex>

/** Es la encargada de enviar las peticiones generadas desde la GUI
 * al servidor.
 *
 */
class ProxyServer: public Proxy {
private:
    /// Morph interno de la clase
    Morph &morph;

    /// Lista de nombres de workspaces
    std::vector<std::string> &workspaces;

    /// Mutex pasado por referencia
    std::mutex &m;

    ///Este flag le indica al proxy que debe ejecutar un comando.
    bool flag;

    /// Mensaje de error si es que hay.
    std::string errorMsg;

private:
    /** Envia el mensaje que esta guardado en message al servidor
     *
     */
    virtual void sendCMDMessage();

public:
    /** Constructor
     * @param socket
     * @papram morph
     * @param workspaces lista de nombres de los workspaces
     * @param m mutex
     */
    ProxyServer(Socket &socket, Morph& morph,
            std::vector<std::string> &workspaces, std::mutex &m);

    /** Envia un mensaje para ejecutar codigo self
     * @param command comando a enviar
     * @param strMessage mensaje a enviar
     */
    bool sendCmdMessage(char command, std::string &strMessage);
```

```
    /** Metodo que sirve para procesar la respuesta que le envia el servidor.
     *
     */
    void run();

    /** Obtiene el flag que determina que envio el comando y
     * recibio la respuesta por parte del servidor. Indica
     * que ya se concretaron las operaciones y que se puede
     * volver a enviar nuevamente.
     */
    bool getFlag() const;

    /** Indica si hubo o no errores informados por el servidor.
     *
     */
    bool areThereErrors() const;

    /** Obtiene los errores. Si no hubo ninguno devuelve una cadena vacio, si
     * hay devuelve el error.
     *
     */
    std::string getErrors();

    /** Setea el flag que indica si se esta esperando una respuesta del servidor
        .
     * @param newValue nuevo valor del flag
     */
    void setFlag(const bool newValue);
};

#endif /* CLIENT_PROXYSERVER_H_ */
```

11 client_proxy_server.cpp

```
#include "client_proxy_server.h"

ProxyServer::ProxyServer(Socket &socket, Morph &morph,
        std::vector<std::string> &workspaces, std::mutex &m) :
        Proxy(socket), morph(morph), workspaces(workspaces), m(m) {
    this->serverSocket.connect();
    setFlag(false);
}

bool ProxyServer::sendCmdMessage(char command, std::string &strMessage) {
    if (getFlag())
        return false;
    else {
        message.setCommand(command);
        message.setMessage(strMessage.c_str());
        message.setLength(strMessage.size());
        setFlag(true);
        return true;
    }
}

void ProxyServer::sendCMDMessage() {
    this->send(message);
}

void ProxyServer::run() {
    while (!_interrupt) {
        if (getFlag()) {
            try {

                //Envio el mensaje al otro extremo de la comunicacion
                sendCMDMessage();

                //Recibo el resultado del mensaje enviado
                int s = receive();
                if (s == 0) {
                    finished = true;
                    break;
                }

                std::string mensajeRecibido;

                switch (message.getCommand()) {
                case ERRORMESSAGE: {
                    errorMsg = message.getMessage();
                    break;
                }
                case OK_MSG_MORPH: {
                    mensajeRecibido = message.getMessage();
                    m.lock();
                    ParserProtocoloMorph(morph, mensajeRecibido);
                    m.unlock();
                    break;
                }
                case OK_MSG_SELECT_WKS: {
```

```
                    mensajeRecibido = message.getMessage();
                    m.lock();
                    ParserProtocoloWorkspaces(workspaces, mensajeRecibido);
                    m.unlock();
                    break;
                }
                default:
                    std::cerr << "Mensaje␣desconocido." << std::endl;
                    break;
                }

            } catch (const std::runtime_error &e) {
                if (!_interrupt)
                    throw e;
            }
            setFlag(false);
        }
    }
}

bool ProxyServer::getFlag() const {
    m.lock();
    bool _flag = flag;
    m.unlock();
    return _flag;
}

void ProxyServer::setFlag(const bool newValue) {
    m.lock();
    flag = newValue;
    m.unlock();
}

bool ProxyServer::areThereErrors() const {
    return (errorMsg.size() > 0);
}

std::string ProxyServer::getErrors() {
    std::string copy = errorMsg;
    errorMsg.clear();
    return copy;
}
```

12 client_column_record.h

```
#include <string>
#include <gtkmm-3.0/gtkmm.h>

/** Esta clase represanta el modelo de columnas que se va
 * a utilizar en el TreeView de los slots.
 *
 */
class ColumnRecord: public Gtk::TreeModel::ColumnRecord {
public:
    ColumnRecord() {
        add(m_col_delete);
        add(m_col_slotName);
        add(m_col_mutable);
        add(m_col_objType);
        add(m_col_preview);
    }

    Gtk::TreeModelColumn<bool> m_col_delete;
    Gtk::TreeModelColumn<Glib::ustring> m_col_slotName;
    Gtk::TreeModelColumn<bool> m_col_mutable;
    Gtk::TreeModelColumn<Glib::ustring> m_col_objType;
    Gtk::TreeModelColumn<Glib::ustring> m_col_preview;
};
```

13 client_column_record_wk.h

```
#ifndef CLIENT_COLUMNRECORDWK_H_
#define CLIENT_COLUMNRECORDWK_H_

#include <string>
#include <gtkmm-3.0/gtkmm.h>

/** Esta clase represanta el modelo de columnas que se va
 * a utilizar en el TreeView que enumera los Workspaces.
 *
 */
class ColumnRecordWk: public Gtk::TreeModel::ColumnRecord {
public:
    ColumnRecordWk() {
        add(m_col_delete);
        add(m_col_wkName);
    }

    Gtk::TreeModelColumn<bool> m_col_delete;
    Gtk::TreeModelColumn<Glib::ustring> m_col_wkName;
};

#endif /* CLIENT_COLUMNRECORDWK_H_ */
```

14 client_parser_protocolo_workspaces.h

```
#ifndef PARSER_PROTOCOLO_WORKSPACES_H_
#define PARSER_PROTOCOLO_WORKSPACES_H_

#include <string>
#include <vector>

/** Esta clase se encarga de parsear los mensajes que lleguen al
 *  cliente con el comando OK_MSG_SELECT_WKS y cargar el vector de
 *  workspaces con la informaci n obtenida.
 */
class ParserProtocoloWorkspaces {
private:
    std::vector<std::string> &workspaces;
    std::string &cad;
    int pCad = 0;

public:
    /** Constructor
     * @param workspaces lista de nombres de workspaces.
     * @param cad cadena que se va a parsear seg n protocolo
     */
    ParserProtocoloWorkspaces(std::vector<std::string> &workspaces,
            std::string &cad);

    /** Constructor por copia deshabilitado
     */
    ParserProtocoloWorkspaces(const ParserProtocoloWorkspaces&) = delete;

    /** Constructor por movimiento deshabilitado
     */
    ParserProtocoloWorkspaces(ParserProtocoloWorkspaces&&) = delete;

    /** Operador de asignacion deshabilitado
     */
    ParserProtocoloWorkspaces& operator=(const ParserProtocoloWorkspaces&) =
        delete;

    /** Operador de asignacion por moviemiento deshabilitado
     */
    ParserProtocoloWorkspaces& operator=(ParserProtocoloWorkspaces&&) = delete;
private:
    /** Captura el siguiente campo de la cadena cad utilizando como separador
     * el caracter especial de protocolo CHAR_SEPARADOR
     */
    std::string getCampo();
};

#endif /* PARSE_RPROTOCOLO_WORKSPACES_H_ */
```

15 client_parser_protocolo_workspaces.cpp

```cpp
#include "client_parser_protocolo_workspaces.h"

#include <iostream>
#include <string>
#include "common_define.h"
#include <vector>

ParserProtocoloWorkspaces::ParserProtocoloWorkspaces(
        std::vector<std::string> &workspaces, std::string &cad) :
        workspaces(workspaces), cad(cad) {
    workspaces.clear();
    std::string wkName;
    workspaces.clear();
    while (pCad < (int) cad.size()) {
        wkName = getCampo();
        pCad++; //Salimos del caracter separador
        workspaces.push_back(wkName);
    }
}

std::string ParserProtocoloWorkspaces::getCampo() {
    std::string campo;
    while (cad[pCad] != CHAR_SEPARADOR and pCad < (int) cad.size()) {
        campo += cad[pCad];
        pCad++;
    }
    return campo;
}
```

16 client_parser_protocolo_morph.h

```
#ifndef PARSER_PROTOCOLO_MORPH_H_
#define PARSER_PROTOCOLO_MORPH_H_

#include <string>
#include <vector>
#include "client_morph.h"

/** Esta clase se encarga de parsear los mensajes que lleguen al
 *  cliente con el comando OK_MSG_MORPH y cargar el objeto Morph
 *  con la informaci n obtenida.
 */
class ParserProtocoloMorph {
private:
    Morph &morph;
    std::string &cad;
    int pCad = 0;

public:
    /** Constructor
     * @param morph Objeto que contiene la informaci n para dibujar
     * en la ventana MorphWindow.
     * @param cad cadena que se va a parsear seg n protocolo
     */
    ParserProtocoloMorph(Morph &morph, std::string &cad);

    /** Constructor por copia deshabilitado
     */
    ParserProtocoloMorph(const ParserProtocoloMorph&) = delete;

    /** Constructor por movimiento deshabilitado
     */
    ParserProtocoloMorph(ParserProtocoloMorph&&) = delete;

    /** Operador de asignacion deshabilitado
     */
    ParserProtocoloMorph& operator=(const ParserProtocoloMorph&) = delete;

    /** Operador de asignacion por moviemiento deshabilitado
     */
    ParserProtocoloMorph& operator=(ParserProtocoloMorph&&) = delete;

private:
    /** Captura el siguiente campo de la cadena cad utilizando como separador
     * el caracter especial de protocolo CHAR_SEPARADOR
     */
    std::string getCampo();
};

#endif /* PARSE_RPROTOCOLO_MORPH_H_ */
```

17 client_parser_protocolo_morph.cpp

```
#include "client_parser_protocolo_morph.h"

#include <iostream>
#include <string>
#include "common_define.h"
#include "client_morph.h"

ParserProtocoloMorph::ParserProtocoloMorph(Morph &morph, std::string &cad) :
        morph(morph), cad(cad) {
    morph.clear();
    std::string objName = getCampo();
    morph.setObjName(objName);
    pCad++; //Salimos del caracter separador

    std::string codeSegment = getCampo();
    morph.setCodeSegment(codeSegment);
    pCad++; //Salimos del caracter separador

    std::string slotName;
    std::string strNativeMethod;
    bool isNativeMethod;
    std::string strMutable;
    bool isMutable;
    std::string strArgument;
    bool isArgument;
    std::string strParent;
    bool isParent;
    std::string objSlotName;
    std::string objSlotPreview;

    while (pCad < (int) cad.size()) {
        slotName = getCampo();
        pCad++; //Salimos del caracter separador

        strNativeMethod = getCampo();
        if (strNativeMethod == FALSE_BIN)
            isNativeMethod = false;
        else
            isNativeMethod = true;
        pCad++; //Salimos del caracter separador

        strMutable = getCampo();
        if (strMutable == FALSE_BIN)
            isMutable = false;
        else
            isMutable = true;
        pCad++; //Salimos del caracter separador

        strParent = getCampo();
        if (strParent == FALSE_BIN)
            isParent = false;
        else
            isParent = true;
        pCad++; //Salimos del caracter separador
```

```
        strArgument = getCampo();
        if (strArgument == FALSE_BIN)
            isArgument = false;
        else
            isArgument = true;
        pCad++; //Salimos del caracter separador

        objSlotName = getCampo();
        pCad++; //Salimos del caracter separador

        objSlotPreview = getCampo();
        pCad++; //Salimos del caracter separador

        morph.addSlot(slotName, isNativeMethod, isMutable, isArgument, isParent,
                objSlotName, objSlotPreview);
    }
}

std::string ParserProtocoloMorph::getCampo() {
    std::string campo;
    while (cad[pCad] != CHAR_SEPARADOR and pCad < (int) cad.size()) {
        campo += cad[pCad];
        pCad++;
    }
    return campo;
}
```

18 client_morph_window.h

```
#ifndef MORPHWINDOW_H_
#define MORPHWINDOW_H_

#include <gtkmm-3.0/gtkmm.h>
#include <gdkmm/color.h>
#include <iostream>
#include <fstream>
#include "client_column_record.h"
#include "client_morph.h"
#include "client_proxy_server.h"

#define COL_DELETE 0
#define COL_MUTABLE 2
#define MAX_PREVIEW_LEN 20

/** Se encarga de dibujar la ventana que representa al Morphic de Self
 * Permite visualizar la informaci n de un objeto de por vez.
 *
 */
class MorphWindow: public Gtk::Window {
private:
    Morph &morph;
    ProxyServer &proxyServer;
    std::mutex &m;

    Glib::RefPtr<Gtk::Builder> refBuilder;

    Gtk::Window *pWindow = nullptr;

    // Botones
    Gtk::Button *pBtnLobby = nullptr;
    Gtk::Button *pBtnGoBack = nullptr;
    Gtk::Button *pBtnRefresh = nullptr;
    Gtk::Button *pBtnEnviar = nullptr;
    Gtk::Button *pBtnApply = nullptr;
    Gtk::Button *pBtnSetSlot = nullptr;
    Gtk::Button *pBtnSetCodeSegment = nullptr;

    // Cajas de texto
    Gtk::TextView *pTxtEntrada = nullptr;
    Gtk::TextView *pTxtCodeSegment = nullptr;
    Gtk::Entry *pTxtObjName = nullptr;
    Gtk::Entry *pTxtSlot = nullptr;

    Gtk::TreeView *pTreeView = nullptr;
    Glib::RefPtr<Gtk::TreeStore> m_refTreeModel;
    ColumnRecord m_Columns;

    // Menu
    Gtk::MenuItem *pMenuItemOpen = nullptr;
    Gtk::MenuItem *pMenuItemCloseWorkspace = nullptr;

private:
    /** Este metodo agrega los widgets a la ventana
     *
```

```
     */
    void addWidgets();

    /** Este metodo configura las columnas los eventos del TreeView.
     *
     */
    void configureTreeView();

    /** Este metodo dibuja los datos recibidos del servidor en el TreeView.
     *
     */
    void drawMorph();

    /** Este metodo sirve para enviar las solicitudes al ProxyServer
     *
     */
    void doAction(char action, std::string text);

    // Eventos
    void btnLobby_clicked();
    void btnGoBack_clicked();
    void btnRefresh_clicked();
    void btnEnviar_clicked();
    void btnApply_clicked();
    void btnSetSlot_clicked();
    void btnSetCodeSegment_clicked();
    bool window_deleted(GdkEventAny* any_event);

    void on_row_activated(const Gtk::TreeModel::Path& path,
            Gtk::TreeViewColumn* column);
    void on_Open_selected();
    void on_CloseWorkspace_selected();
    void cellMutable_toggled(const Glib::ustring &path);
    void cellDelete_toggled(const Glib::ustring &path);
public:
    /** Constructor de la clase
     * @param morph referencia al objeto Morph.
     * @param proxyServer referencia al proxy
     * @param m referencia al mutex
     */
    MorphWindow(Morph &morph, ProxyServer &proxyServer, std::mutex &m);

    /** Destructor de la clase
     *
     */
    ~MorphWindow();

    /** Devuelve un puntero al objeto Gtk::Window.
     *
     */
    Gtk::Window *getWindow();

    // Elimino constructores y operadores de asignacion
    MorphWindow(MorphWindow&&) = delete;
    MorphWindow(const MorphWindow&) = delete;

    MorphWindow& operator=(MorphWindow&&) = delete;
```

```
    MorphWindow& operator=(const MorphWindow&) = delete;
};

#endif /* MORPHWINDOW_H_ */
```

19 client_morph_window.cpp

```
#include "client_morph_window.h"

MorphWindow::MorphWindow(Morph &morph, ProxyServer &proxyServer, std::mutex &m)
    :
        morph(morph), proxyServer(proxyServer), m(m) {
    refBuilder = Gtk::Builder::create();
    try {
        refBuilder->add_from_file(GLADE_FILE);
    } catch (...) {
        throw std::runtime_error("No se puede crear la ventana");
    }

    refBuilder->get_widget("window", pWindow);

    addWidgets();
    configureTreeView();
    drawMorph();
    pWindow->signal_delete_event().connect(
            sigc::mem_fun(*this, &MorphWindow::window_deleted));
    show_all_children();
}

MorphWindow::~MorphWindow() {
    delete pWindow;
}

void MorphWindow::addWidgets() {
    // Primera fila
    refBuilder->get_widget("btnLobby", pBtnLobby);
    refBuilder->get_widget("btnGoBack", pBtnGoBack);
    refBuilder->get_widget("btnRefresh", pBtnRefresh);
    refBuilder->get_widget("btnEnviar", pBtnEnviar);
    refBuilder->get_widget("txtEntrada", pTxtEntrada);

    // Segunda fila
    refBuilder->get_widget("btnApply", pBtnApply);
    refBuilder->get_widget("txtObjectName", pTxtObjName);

    refBuilder->get_widget("treeView", pTreeView);
    refBuilder->get_widget("btnSetSlot", pBtnSetSlot);
    refBuilder->get_widget("txtSlot", pTxtSlot);

    refBuilder->get_widget("btnSetCodeSegment", pBtnSetCodeSegment);
    refBuilder->get_widget("txtCodeSegment", pTxtCodeSegment);

    refBuilder->get_widget("m_Open", pMenuItemOpen);
    refBuilder->get_widget("m_CloseWorkspace", pMenuItemCloseWorkspace);

    if (pBtnLobby) {
        pBtnLobby->signal_clicked().connect(
                sigc::mem_fun(*this, &MorphWindow::btnLobby_clicked));
    }

    if (pBtnGoBack) {
        pBtnGoBack->signal_clicked().connect(
```

```
                sigc::mem_fun(*this, &MorphWindow::btnGoBack_clicked));
    }

    if (pBtnRefresh) {
        pBtnRefresh->signal_clicked().connect(
                sigc::mem_fun(*this, &MorphWindow::btnRefresh_clicked));
    }

    if (pBtnEnviar) {
        pBtnEnviar->signal_clicked().connect(
                sigc::mem_fun(*this, &MorphWindow::btnEnviar_clicked));
    }

    if (pBtnApply) {
        pBtnApply->signal_clicked().connect(
                sigc::mem_fun(*this, &MorphWindow::btnApply_clicked));
    }

    if (pBtnSetSlot) {
        pBtnSetSlot->signal_clicked().connect(
                sigc::mem_fun(*this, &MorphWindow::btnSetSlot_clicked));
    }

    if (pBtnSetCodeSegment) {
        pBtnSetCodeSegment->signal_clicked().connect(
                sigc::mem_fun(*this, &MorphWindow::btnSetCodeSegment_clicked));
    }

    if (pTreeView) {
        pTreeView->signal_row_activated().connect(
                sigc::mem_fun(*this, &MorphWindow::on_row_activated));
    }

    if (pMenuItemOpen) {
        pMenuItemOpen->signal_activate().connect(
                sigc::mem_fun(*this, &MorphWindow::on_Open_selected));
    }

    if (pMenuItemCloseWorkspace) {
        pMenuItemCloseWorkspace->signal_activate().connect(
                sigc::mem_fun(*this, &MorphWindow::on_CloseWorkspace_selected));
    }
}

void MorphWindow::configureTreeView() {
    m_refTreeModel = Gtk::TreeStore::create(m_Columns);
    pTreeView->set_model(m_refTreeModel);

    // Creo las columnas del TreeView
    pTreeView->append_column_editable("[␣-␣]", m_Columns.m_col_delete);
    pTreeView->append_column("Slot", m_Columns.m_col_slotName);
    pTreeView->append_column_editable("Mutable", m_Columns.m_col_mutable);
    pTreeView->append_column("Objeto", m_Columns.m_col_objType);
    pTreeView->append_column("Vista␣Previa", m_Columns.m_col_preview);

    // Le asigno el evento toggle a las columnas que tienen un checkbox
    ((Gtk::CellRendererToggle*) pTreeView->get_column_cell_renderer(COL_DELETE))
```

```
        ->signal_toggled().connect(
            sigc::mem_fun(*this, &MorphWindow::cellDelete_toggled));

    ((Gtk::CellRendererToggle*) pTreeView->get_column_cell_renderer(COL_MUTABLE)
        )->signal_toggled().connect(
            sigc::mem_fun(*this, &MorphWindow::cellMutable_toggled));
}

void MorphWindow::drawMorph() {
    m.lock();
    pTxtObjName->set_text(morph.getObjName());

    m_refTreeModel->clear();
    Gtk::TreeModel::Row row;

    for (int i = 0; i < morph.getSlotsSize(); i++) {
        row = *(m_refTreeModel->append());
        std::string slotName;
        if (morph.isArgumentSlot(i))
            slotName = OP_ARG + morph.getSlotName(i);
        else if (morph.isParentSlot(i))
            slotName = morph.getSlotName(i) + OP_PARENT;
        else
            slotName = morph.getSlotName(i);

        std::string preview = morph.getSlotObjPreview(i);

        if (preview.size() > MAX_PREVIEW_LEN)
            preview = preview.substr(0, MAX_PREVIEW_LEN);

        row[m_Columns.m_col_slotName] = slotName;
        row[m_Columns.m_col_mutable] = morph.isMutableSlot(i);
        row[m_Columns.m_col_objType] = morph.getSlotObjName(i);
        row[m_Columns.m_col_preview] = preview;
    }
    auto pTextBuffer = Glib::RefPtr < Gtk::TextBuffer
            > ::cast_dynamic(refBuilder->get_object("textBufferCodeSegment"));
    pTextBuffer->set_text(morph.getCodeSegment());
    m.unlock();
}

Gtk::Window* MorphWindow::getWindow() {
    return pWindow;
}

void MorphWindow::doAction(char action, std::string text) {
    proxyServer.sendCmdMessage(action, text);
    while (proxyServer.getFlag()) {
    }
    if (proxyServer.areThereErrors()) {
        Gtk::MessageDialog dialog(*this, "Errores␣en␣la␣ejecuci n", false,
                Gtk::MESSAGE_ERROR, Gtk::BUTTONS_OK);
        dialog.set_secondary_text(proxyServer.getErrors());
        dialog.run();
    } else {
        drawMorph();
    }
```

```
}

// Eventos
bool MorphWindow::window_deleted(GdkEventAny* any_event) {
    Gtk::MessageDialog dialog(*this, " Abandonar?", false,
            Gtk::MESSAGE_QUESTION, Gtk::BUTTONS_YES_NO);
    dialog.set_secondary_text(" Desea␣abandonar␣el␣Workspace?");
    Gtk::ResponseType resp = (Gtk::ResponseType) dialog.run();

    if (resp == Gtk::RESPONSE_NO) {
        return true;
    }

    std::string emptyString = "";
    proxyServer.sendCmdMessage(CLOSE_WK, emptyString);
    while (proxyServer.getFlag()) {
    }

    return false;
}

void MorphWindow::btnEnviar_clicked() {
    if (pWindow) {
        auto pTextBuffer = Glib::RefPtr < Gtk::TextBuffer
                > ::cast_dynamic(refBuilder->get_object("txtBufferEntrada"));
        std::string text = pTextBuffer->get_text();
        doAction(EXEC_LOBBY_CMD, text);
        pTextBuffer->set_text("");
    }
}

void MorphWindow::btnLobby_clicked() {
    std::string empty;
    doAction(SHOW_LOBBY, empty);
}

void MorphWindow::btnGoBack_clicked() {
    std::string empty;
    doAction(GO_BACK, empty);
}
void MorphWindow::btnRefresh_clicked() {
    std::string empty;
    doAction(EXEC_REFRESH, empty);
}

void MorphWindow::btnApply_clicked() {
    std::string text = pTxtObjName->get_text();
    doAction(SET_OBJ_NAME, text);
    pTxtObjName->set_text("");
}

void MorphWindow::btnSetSlot_clicked() {
    std::string text = pTxtSlot->get_text();
    doAction(ADD_SLOT, text);
    pTxtSlot->set_text("");
}
```

```
void MorphWindow::btnSetCodeSegment_clicked() {
    auto pTextBuffer = Glib::RefPtr < Gtk::TextBuffer
            > ::cast_dynamic(refBuilder->get_object("textBufferCodeSegment"));
    std::string text = pTextBuffer->get_text();

    doAction(SET_CODESEGMENT, text);
}

void MorphWindow::on_row_activated(const Gtk::TreeModel::Path& path,
        Gtk::TreeViewColumn* column) {
    Gtk::TreeModel::iterator iter = m_refTreeModel->get_iter(path);
    if (iter) {
        int rowId = path[0];

        Glib::ustring text = morph.getSlotName(rowId);
        std::string str(text.c_str());

        if (morph.isNativeMethodSlot(rowId)) {
            return;
        }

        doAction(GET_SLOT_OBJ, str);
    }
}

void MorphWindow::on_CloseWorkspace_selected() {
    this->getWindow()->close();
}

void MorphWindow::on_Open_selected() {
    Gtk::FileChooserDialog dialog("Seleccionar un archivo para procesar",
            Gtk::FILE_CHOOSER_ACTION_OPEN);
    dialog.set_transient_for(*pWindow);

    // Le agrego los botones de Abrir y Cancelar al cuadro de dialogo
    dialog.add_button("_Cancelar", Gtk::RESPONSE_CANCEL);
    dialog.add_button("_Abrir", Gtk::RESPONSE_OK);

    auto filter_any = Gtk::FileFilter::create();
    filter_any->set_name("Archivos self");
    filter_any->add_pattern("*.self");
    dialog.add_filter(filter_any);

    int result = dialog.run();

    switch (result) {
    case (Gtk::RESPONSE_OK): {

        std::string filename = dialog.get_filename();
        std::ifstream file(filename);

        std::stringstream buffer;
        buffer << file.rdbuf();

        auto pTextBuffer = Glib::RefPtr < Gtk::TextBuffer
                > ::cast_dynamic(refBuilder->get_object("txtBufferEntrada"));
        pTextBuffer->set_text(buffer.str());
```

```
            break;
        }
        case (Gtk::RESPONSE_CANCEL): {
            break;
        }
        default:
            break;
        }
    }

    void MorphWindow::cellMutable_toggled(const Glib::ustring &path) {
        Gtk::TreeIter iter;
        auto model = this->m_refTreeModel;
        if (model) {
            iter = model->get_iter(path);

            bool checked = iter->get_value(m_Columns.m_col_mutable);
            std::string text = iter->get_value(m_Columns.m_col_slotName);

            int rowId = atoi(path.c_str());
            // Antes de enviar el comando al servidor, verifico que sea objeto
                nativo
            if (morph.isNativeMethodSlot(rowId)) {
                iter->set_value(m_Columns.m_col_mutable, false);
                return;
            }

            proxyServer.sendCmdMessage(SWAP_MUTABILITY, text);

            while (proxyServer.getFlag()) {
            }

            if (proxyServer.areThereErrors()) {
                Gtk::MessageDialog dialog(*this, "Errores␣en␣la␣ejecuci n", false,
                        Gtk::MESSAGE_ERROR, Gtk::BUTTONS_OK);
                dialog.set_secondary_text(proxyServer.getErrors());
                dialog.run();
                iter->set_value(m_Columns.m_col_mutable, checked);
            }
        }
    }

    void MorphWindow::cellDelete_toggled(const Glib::ustring &path) {
        Gtk::TreeIter iter;
        auto model = this->m_refTreeModel;
        if (model) {
            iter = model->get_iter(path);
            int rowId = atoi(path.c_str());

            std::string slotName = morph.getSlotName(rowId);

            Gtk::MessageDialog dialog(*this, "Confirmar", false,
                    Gtk::MESSAGE_QUESTION, Gtk::BUTTONS_YES_NO);
            dialog.set_secondary_text(" Desea␣borrar␣el␣slot␣" + slotName + "?");
            Gtk::ResponseType resp = (Gtk::ResponseType) dialog.run();

            if (resp == Gtk::RESPONSE_NO) {
```

```
            iter->set_value(m_Columns.m_col_delete, false);
            return;
        }

        proxyServer.sendCmdMessage(REMOVE_SLOT, slotName);
        while (proxyServer.getFlag()) {
        }
        if (proxyServer.areThereErrors()) {
            Gtk::MessageDialog dialog(*this, "Errores en la ejecuci n", false,
                    Gtk::MESSAGE_ERROR, Gtk::BUTTONS_OK);
            dialog.set_secondary_text(proxyServer.getErrors());
            dialog.run();
            iter->set_value(m_Columns.m_col_delete, false);
        } else {
            drawMorph();
        }
    }
}
```

20 client_select_wk_window.h

```
#ifndef CLIENT_SELECTWKWINDOW_H_
#define CLIENT_SELECTWKWINDOW_H_

#include <gtkmm-3.0/gtkmm.h>
#include <gdkmm/color.h>
#include <iostream>
#include <mutex>
#include "client_column_record_wk.h"
#include "client_morph_window.h"

/** Se encarga de dibujar la ventana selectora de workspaces.
 *
 */
class SelectWkWindow: public Gtk::Window {
private:
    Morph &morph;
    std::vector<std::string> &workspaces;
    ProxyServer &proxyServer;
    std::mutex &m;

    Glib::RefPtr<Gtk::Builder> refBuilder;

    Gtk::Window *pWindow = nullptr;

    // Botones
    Gtk::Button *pBtnRefreshWk = nullptr;
    Gtk::Button *pBtnNewWk = nullptr;

    // Cajas de texto
    Gtk::Entry *pTxtNewWk = nullptr;

    // TreeView
    Gtk::TreeView *pTreeView = nullptr;
    Glib::RefPtr<Gtk::TreeStore> m_refTreeModel;
    ColumnRecordWk m_Columns;

private:
    //Eventos
    void btnRefreshWk_clicked();
    void btnNewWk_clicked();
    void treeView_toggled(const Glib::ustring &path);
    void treeView_on_row_activated(const Gtk::TreeModel::Path& path,
            Gtk::TreeViewColumn* column);

    /** Este metodo agrega los widgets a la ventana
     *
     */
    void addWidgets();

    /** Este metodo configura las columnas los eventos del TreeView.
     *
     */
    void configureTreeView();

    /** Este metodo dibuja los datos recibidos sobre los workspaces
```

```
     * provenientes del servidor en el TreeView.
     * Previamente hay que llamar al metodo updateList() para actualizar
     * los datos en memoria. Este metodo no los actualiza, solo dibuja lo que
     * esta cargado.
     *
     */
    void drawWorkspaces();

    /** Manda la solicitud al servidor para actualizar la lista
     * de workspaces disponibles que esta guardada en memoria.
     * Para redibujar la lista, llamar a drawWorkspaces().
     */
    void updateList();

public:
    /** Constructor de la clase
     * @param morph referencia al objeto Morph.
     * @param workspaces referencia al vector con los nombres de los workspaces
     * @param proxyServer referencia al proxy
     * @param m referencia al mutex
     */
    SelectWkWindow(Morph &morph, std::vector<std::string> &workspaces,
            ProxyServer &proxyServer, std::mutex &m);

    /** Devuelve un puntero al objeto Gtk::Window.
     *
     */
    Gtk::Window *getWindow();

    // Elimino los contructores por copia y moviemiento. Tambien los
    // operadores de asignacion
    SelectWkWindow(const SelectWkWindow&) = delete;
    SelectWkWindow(SelectWkWindow&&) = delete;
    SelectWkWindow& operator=(const SelectWkWindow&) = delete;
    SelectWkWindow& operator=(SelectWkWindow&&) = delete;
};

#endif /* CLIENT_SELECTWKWINDOW_H_ */
```

21 client_select_wk_window.cpp

```
#include "client_select_wk_window.h"

SelectWkWindow::SelectWkWindow(Morph &morph,
        std::vector<std::string> &workspaces, ProxyServer &proxyServer,
        std::mutex &m) :
        morph(morph), workspaces(workspaces), proxyServer(proxyServer), m(m) {
    refBuilder = Gtk::Builder::create();
    try {
        refBuilder->add_from_file(GLADE_FILE);
    } catch (...) {
        throw std::runtime_error("No se puede crear la ventana");
    }

    refBuilder->get_widget("windowWorkspace", pWindow);

    std::string cad = "";
    proxyServer.sendCmdMessage(AVAILABLE_WKS, cad);
    while (proxyServer.getFlag()) {
    }
    addWidgets();
    configureTreeView();
    drawWorkspaces();
    show_all_children();
}

Gtk::Window* SelectWkWindow::getWindow() {
    return pWindow;
}

void SelectWkWindow::addWidgets() {
    // Primera fila
    refBuilder->get_widget("btnRefreshWk", pBtnRefreshWk);
    refBuilder->get_widget("btnNewWk", pBtnNewWk);
    refBuilder->get_widget("txtNewWk", pTxtNewWk);

    // TreeView
    refBuilder->get_widget("treeViewWks", pTreeView);

    // Conecto los eventos
    if (pBtnRefreshWk) {
        pBtnRefreshWk->signal_clicked().connect(
                sigc::mem_fun(*this, &SelectWkWindow::btnRefreshWk_clicked));
    }

    if (pBtnNewWk) {
        pBtnNewWk->signal_clicked().connect(
                sigc::mem_fun(*this, &SelectWkWindow::btnNewWk_clicked));
    }

    if (pTreeView) {
        pTreeView->signal_row_activated().connect(
                sigc::mem_fun(*this,
                        &SelectWkWindow::treeView_on_row_activated));
    }
```

```
}

void SelectWkWindow::configureTreeView() {
    m_refTreeModel = Gtk::TreeStore::create(m_Columns);
    pTreeView->set_model(m_refTreeModel);

    pTreeView->append_column_editable("[␣-␣]", m_Columns.m_col_delete);
    pTreeView->append_column("Workspace", m_Columns.m_col_wkName);

    // Conecto el evento para controlar el checkbox
    ((Gtk::CellRendererToggle*) pTreeView->get_column_cell_renderer(0))->
        signal_toggled().connect(
            sigc::mem_fun(*this, &SelectWkWindow::treeView_toggled));
}

void SelectWkWindow::drawWorkspaces() {
    m_refTreeModel->clear();
    for (auto str : workspaces) {
        Gtk::TreeModel::Row row = *(m_refTreeModel->append());
        row[m_Columns.m_col_delete] = false;
        row[m_Columns.m_col_wkName] = str;
    }
}

void SelectWkWindow::updateList() {
    std::string empty = "";
    proxyServer.sendCmdMessage(AVAILABLE_WKS, empty);
    while (proxyServer.getFlag()) {
    }
}

// Eventos
void SelectWkWindow::btnRefreshWk_clicked() {
    updateList();
    drawWorkspaces();
}

void SelectWkWindow::btnNewWk_clicked() {
    std::string wkName = pTxtNewWk->get_text();

    if (wkName.size() == 0) {
        Gtk::MessageDialog dialog(*this, "Errores␣en␣la␣ejecuci n", false,
                Gtk::MESSAGE_ERROR, Gtk::BUTTONS_OK);
        dialog.set_secondary_text(
                "No␣se␣indico␣un␣nombre␣para␣el␣nuevo␣Workspace.");
        dialog.run();
        updateList();
    } else if (wkName.find(CHAR_SEPARADOR) != std::string::npos) {
        Gtk::MessageDialog dialog(*this, "Errores␣en␣la␣ejecuci n", false,
                Gtk::MESSAGE_ERROR, Gtk::BUTTONS_OK);
        std::stringstream ss;
        std::string s;
        char c = CHAR_SEPARADOR;
        ss << c;
        ss >> s;
        dialog.set_secondary_text(
                "El␣caracter␣especial␣" + s + "␣est ␣prohibido␣por␣protocolo.")
```

```
                    ;
            dialog.run();
            updateList();
        } else {

            proxyServer.sendCmdMessage(NEW_WK, wkName);
            while (proxyServer.getFlag()) {
            }

            if (proxyServer.areThereErrors()) {
                Gtk::MessageDialog dialog(*this, "Errores en la ejecuci n", false,
                        Gtk::MESSAGE_ERROR, Gtk::BUTTONS_OK);
                dialog.set_secondary_text(
                        proxyServer.getErrors() + '\n'
                                + "Se actualizara la lista.");
                dialog.run();
                updateList();
            }
        }
        drawWorkspaces();
        pTxtNewWk->set_text("");
    }

    void SelectWkWindow::treeView_toggled(const Glib::ustring &path) {
        Gtk::TreeIter iter;

        auto model = this->m_refTreeModel;
        if (model) {
            iter = model->get_iter(path);

            std::string wkName = iter->get_value(m_Columns.m_col_wkName);
            Gtk::MessageDialog dialog(*this, " Esta seguro?", false,
                    Gtk::MESSAGE_QUESTION, Gtk::BUTTONS_YES_NO);
            dialog.set_secondary_text(" Desea borrar el Workspace " + wkName + "?")
                    ;
            Gtk::ResponseType resp = (Gtk::ResponseType) dialog.run();

            if (resp == Gtk::RESPONSE_NO) {
                iter->set_value(m_Columns.m_col_delete, false);
                updateList();
            } else {

                proxyServer.sendCmdMessage(DELETE_WK, wkName);
                while (proxyServer.getFlag()) {
                }
                if (proxyServer.areThereErrors()) {
                    Gtk::MessageDialog dialog(*this, "Errores en la ejecuci n",
                            false, Gtk::MESSAGE_ERROR, Gtk::BUTTONS_OK);
                    dialog.set_secondary_text(proxyServer.getErrors());
                    dialog.run();
                    updateList();
                }
            }
            drawWorkspaces();
        }
    }
```

```
void SelectWkWindow::treeView_on_row_activated(const Gtk::TreeModel::Path& path,
        Gtk::TreeViewColumn* column) {
    Gtk::TreeModel::iterator iter = m_refTreeModel->get_iter(path);
    if (iter) {
        Gtk::TreeModel::Row row = *iter;
        Glib::ustring text = row[m_Columns.m_col_wkName];
        std::string sText = std::string(text.c_str());

        proxyServer.sendCmdMessage(LOAD_WK, sText);
        while (proxyServer.getFlag()) {
        }

        // Creo la ventana con los morphs y la muestro.
        MorphWindow* _window = new MorphWindow(morph, proxyServer, m);
        _window->getWindow()->show();
    }
}
```

22 client_morph.h

```
#ifndef CLIENT_MORPH_H_
#define CLIENT_MORPH_H_

#include <string>
#include <tuple>
#include <vector>

/** Este objeto contiene la informaci n que ve el cliente desde la GUI
 * y representa al objeto que est  visualizando.
 */
class Morph {
public:
    /** Slot del objeto Morph.
     * Elementos de slot_t
     * pos          tipo                variable \n
     * 0            std::string nombreSlot \n
     * 1            bool                esMetodoNativo \n
     * 2            bool                esMutable \n
     * 3            bool                esArgument \n
     * 4            bool                esParent \n
     * 5            std::string nombreObjSlot \n
     * 6            std::string previewObjSlot
     * */
    typedef std::tuple<std::string, bool, bool, bool, bool, std::string,
            std::string> slot_morph;

private:
    std::string objName;
    std::string codeSegment;
    std::vector<slot_morph> slots;

public:
    /** Constructor
     */
    Morph();
    /** Constructor por copia deshabilitado
     */
    Morph(const Morph&) = delete;
    /** Constructor por movimiento deshabilitado
     */
    Morph(Morph&&) = delete;
    /** Operador de asignacion deshabilitado
     */
    Morph& operator=(const Morph&) = delete;
    /** Operador de asignacion por moviemiento deshabilitado
     */
    Morph& operator=(Morph&&) = delete;
    /** Limpia el contenido del morph
     */
    void clear();

    //Getters
    /** Retorna el nombre del objeto.
     */
    std::string getObjName() const;
```

```
/** Retorna el bloque de c d i g o del objeto.
 */
std::string getCodeSegment() const;
/** Retorna la cantidad de slots del objeto.
 */
int getSlotsSize() const;
/** Retorna el nombre del slot indicado.
 * @param nSlot n m e r o de slot del que se solicita informaci n.
 */
std::string getSlotName(int nSlot) const;
/** Retorna si el slot indicado es un m t o d o nativo.
 * @param nSlot n m e r o de slot del que se solicita informaci n.
 */
bool isNativeMethodSlot(int nSlot) const;
/** Retorna si el slot indicado es mutable.
 * @param nSlot n m e r o de slot del que se solicita informaci n.
 */
bool isMutableSlot(int nSlot) const;
/** Retorna si el slot indicado es de tipo argumento.
 * @param nSlot n m e r o de slot del que se solicita informaci n.
 */
bool isArgumentSlot(int nSlot) const;
/** Retorna si el slot indicado es de tipo parent.
 * @param nSlot n m e r o de slot del que se solicita informaci n.
 */
bool isParentSlot(int nSlot) const;
/** Retorna el nombre del objeto contenido en el slot indicado.
 * @param nSlot n m e r o de slot del que se solicita informaci n.
 */
std::string getSlotObjName(int nSlot) const;
/** Retorna una cadena con la vista previa del objeto contenido
 * en el slot indicado.
 * @param nSlot n m e r o de slot del que se solicita informaci n.
 */
std::string getSlotObjPreview(int nSlot) const;

//Setters
/** Setea el nombre del objeto
 * @param cad nuevo nombre del objeto.
 */
void setObjName(std::string &cad);
/** Setea el bloque de c d i g o del objeto
 * @param cad nuevo bloque de c d i g o del objeto.
 */
void setCodeSegment(std::string &cad);
/** Agrega un slot al Objeto Morph
 * @param slotName nombre del slot.
 * @param isNativeMethod indica si es un m t o d o nativo.
 * @param isMutable indica si el slot es mutable.
 * @param isArgument indica si el slot es de tipo argumento.
 * @param isParent indica si el slot es de tipo parent.
 * @param objSlotName nombre del objeto contenido en el slot.
 * @param objSlotPreview vista previa del objeto contenido en el slot.
 */
void addSlot(std::string &slotName, bool isNativeMethod, bool isMutable,
        bool isArgument, bool isParent, std::string &objSlotName,
        std::string &objSlotPreview);
```

```
    /** Imprime por pantalla los campos del Morph
     */
    void mostrar();
};

#endif /* CLIENT_MORPH_H_ */
```

23 client_morph.cpp

```cpp
#include "client_morph.h"

#include <string>
#include <iostream>
#include <tuple>
#include <vector>

Morph::Morph() {
    clear();
}

void Morph::clear() {
    objName = std::string("ObjectName");
    codeSegment = std::string("codeSegment..");
    //Hay que verificar que no queden cosas colgadas al hacer el clear del
        vector
    slots = std::vector<slot_morph> { };
}

std::string Morph::getObjName() const {
    return objName;
}

std::string Morph::getCodeSegment() const {
    return codeSegment;
}

int Morph::getSlotsSize() const {
    return slots.size();
}

std::string Morph::getSlotName(int nSlot) const {
    return std::get<0>(slots.at(nSlot));
}

bool Morph::isNativeMethodSlot(int nSlot) const {
    return std::get<1>(slots.at(nSlot));
}

bool Morph::isMutableSlot(int nSlot) const {
    return std::get<2>(slots.at(nSlot));
}

bool Morph::isArgumentSlot(int nSlot) const {
    return std::get<3>(slots.at(nSlot));
}

bool Morph::isParentSlot(int nSlot) const {
    return std::get<4>(slots.at(nSlot));
}

std::string Morph::getSlotObjName(int nSlot) const {
    return std::get<5>(slots.at(nSlot));
}
```

```
std::string Morph::getSlotObjPreview(int nSlot) const {
    return std::get<6>(slots.at(nSlot));
}

void Morph::setObjName(std::string &cad) {
    objName = cad;
}

void Morph::setCodeSegment(std::string &cad) {
    codeSegment = cad;
}

void Morph::addSlot(std::string &slotName, bool isNativeMethod, bool isMutable,
        bool isArgument, bool isParent, std::string &objSlotName,
        std::string &objSlotPreview) {
    slot_morph slot = std::make_tuple(slotName, isNativeMethod, isMutable,
            isArgument, isParent, objSlotName, objSlotPreview);
    slots.push_back(slot);
}

void Morph::mostrar() {
    std::cout << "MORPH" << std::endl;
    std::cout << "ObjName: " << objName << std::endl;
    std::cout << "CodeSegment: " << codeSegment << std::endl;

    int i;
    for (i = 0; i < getSlotsSize(); i++) {
        std::cout << std::endl;
        std::cout << "N Slot: " << std::to_string(i) << std::endl;
        std::cout << "SlotName: " << getSlotName(i) << std::endl;
        std::cout << "isNativeMethod: " << isNativeMethodSlot(i) << std::endl;
        std::cout << "isMutable: " << isMutableSlot(i) << std::endl;
        std::cout << "isArgument: " << isArgumentSlot(i) << std::endl;
        std::cout << "isParent: " << isParentSlot(i) << std::endl;
        std::cout << "objSlotName: " << getSlotObjName(i) << std::endl;
        std::cout << "objSlotPreview: " << getSlotObjPreview(i) << std::endl;
    }
}
```

24 server_main.cpp

```
#include <iostream>
#include <fstream>

#include "server_accepter.h"
#include "server_server.h"

#define SERVER_PARAMS 3
const char* SERVER_MODE_SERVER = "s";
const char* SERVER_MODE_FILE = "f";
#define RET_OK 0
#define RET_NOK 1

#include "server_mode_selector.h"

int main(int argc, char **argv) {
    try {
        if (argc == SERVER_PARAMS) {
            char* mode = *(argv + 1);
            if (strcmp(mode, SERVER_MODE_SERVER) == 0) {
                int port = 0;
                try {
                    port = std::stoi(*(argv + 2));
                } catch (...) {
                    throw std::runtime_error("puerto invalido.");
                }
                ModeSelector modeSelector(port);
            } else if (strcmp(mode, SERVER_MODE_FILE) == 0) {
                std::string filename = std::string(*(argv + 2));
                ModeSelector modeSelector(filename);
            } else {
                std::cerr << "Forma de uso: >>> ./server <modo s|f> <port|file>"
                        << std::endl;
            }
        } else {
            std::cerr << "Forma de uso: >>> ./server <modo s|f> <port|file>"
                    << std::endl;
            return RET_NOK;
        }
    } catch (const std::runtime_error &e) {
        std::cout << "Error. " << std::endl << e.what() << std::endl;
    } catch (...) {
        std::cout << "Error desconocido." << std::endl << std::endl;
    }
    return RET_OK;
}
```

25 server_workspace.h

```
#ifndef SERVER_WORKSPACE_H_
#define SERVER_WORKSPACE_H_

#include "server_object.h"
#include "server_parser.h"
#include "server_virtual_machine.h"
#include <string>
#include <iostream>

/** Representa un ambiente de trabajo (workspace) y es el encargado de crear
 * el lobby y de llamar al parser para ejecutar los scripts de c digo self.
 *
 */
class Workspace {
private:
    Object *lobby;
    VirtualMachine vm;

public:
    /// Constructor
    Workspace();

    Workspace(const Workspace&) = delete;
    Workspace(Workspace&&) = delete;

    Workspace& operator=(const Workspace&) = delete;
    Workspace& operator=(Workspace&&) = delete;

    /// Destructor
    ~Workspace();

    /** Recibe codigo self y lo ejecuta.
     * @param code [std::string] codigo que recibe
     * @return Devuelve un Object* con el resultado de la ejecucion
     *
     */
    uint32_t receive(Object* context, std::string &code);

    /// Devuelve el objeto lobby.
    Object* getLobby();

    /** Busca el objeto por ID y retorna el objeto real.
     * @param id id del objeto.
     *
     */
    Object* findObjectById(uint32_t id);
};

#endif /* SERVER_WORKSPACE_H_ */
```

26 server_workspace.cpp

```
#include "server_workspace.h"
#include "common_define.h"

Workspace::Workspace() {
    lobby = new Object();
    std::string _lobby = LOBBY;
    lobby->setName(_lobby);
    //El slot lobby es necesario para cuando se realizan llamadas explicitas a
        lobby.
    lobby->addSlot(_lobby, lobby, false, true, false);
    lobby->enableNativeMethod("collect");
    vm.setLobby(lobby);
}

Workspace::~Workspace() {
    lobby->collect(std::vector<Object*> { });
    delete lobby;
}

uint32_t Workspace::receive(Object* context, std::string &code) {
    Parser parser(vm, context);
    std::vector<Object*> objs = parser.parse(code);
    int size = objs.size() - 1;
    if (size >= 0) {
        return objs[size]->getId();
    } else
        throw std::runtime_error("Error␣de␣sintaxis");
}

Object* Workspace::getLobby() {
    return lobby;
}

Object* Workspace::findObjectById(uint32_t id) {
    return vm.findObjectById(id);
}
```

27 server_parser.h

```
#ifndef SERVER_PARSER_H_
#define SERVER_PARSER_H_

#include <string>

#include "server_object.h"
#include "server_virtual_machine.h"

/** Es la clase encargada de parsear scripts en c digo self y de indicarle
 * a la maquina virtual (VM) que objetos y mensajes se deben emitir
 * en consecuencia.
 */
class Parser {
private:
    /// Objeto sobre el que se ejecuta el script.
    Object* context;
    /// String del script a ejecutar
    std::string cad;
    /// Posicion de la cadena cad sobre la que est  analizando el parser.
    uint32_t pCad;
    /// Modo debug
    bool debug = false;

    /** Cuando vale 1 se ejecuta el metodo receiveMessage
     * Cada vez que se ingresa a un nuevo script incrementa en 1.
     */
    int flagExecute;
    /// M quina virtual del workspace
    VirtualMachine &vm;

public:
    /** Constructor
     * @param vm M quina virtual
     * @param context Objeto sobre el que se ejecuta el script.
     *
     */
    Parser(VirtualMachine &vm, Object* context);

    /** Constructor por copia deshabilitado
     */
    Parser(const Parser&) = delete;
    /** Constructor por movimiento deshabilitado
     */
    Parser(Parser&&) = delete;
    /** Operador de asignacion deshabilitado
     */
    Parser& operator=(const Parser&) = delete;
    /** Operador de asignacion por moviemiento deshabilitado
     */
    Parser& operator=(Parser&&) = delete;

    /** Inicia la secuencia de parseo del script en c digo self
     * @param cad c digo self
     */
    std::vector<Object*> parse(std::string &cad);
```

```
private:
    /** Esta es la funci n  m s  importante del parser, se encarga de decidir
     * que objetos deben ser clonados antes de aplicarles un recv_message.
     * Para ello valida si son:
     * data objects: (objetos primivitos o que tengan codeSegment vacio).
     * method objects: son aquellos que no son data objects e incluye a los
     * metodos nativos.
     * Los m todos nativos son un caso especial ya que alg nos requieren de la
     * clonaci n como los operadores binarios y hay otros que no la requieren.
     *
     */

    Object* receiveMessage(Object* obj, std::string &strName,
            std::vector<Object*> &args);

    /// Valida si en la cadena continua un script
    std::vector<Object*> script();
    /// Valida si en la cadena continua una expression
    Object* expression();
    /// Valida si en la cadena continua una expressionCP
    Object* expressionCP();
    /// Valida si en la cadena continua una expressionP
    Object* expressionP();
    /// Valida si en la cadena continua un keyword message
    Object* keywordMessage();
    /// Valida si en la cadena continua un binary message
    Object* binaryMessage();
    /// Valida si en la cadena continua un unary message
    Object* unaryMessage();
    /// Valida si en la cadena continua un receiver
    Object* receiver();
    /// Valida si en la cadena continua un slotList
    bool slotList(Object* objContenedor);
    /// Valida si en la cadena continua un slotNameExtended
    bool slotNameExtended(int &tipoSlot, std::string &strName);
    /// Valida si en la cadena continua un constant
    Object* constant();
    /// Valida si en la cadena continua un operador
    bool operador(std::string &strOperador);
    /// Valida si en la cadena continua un operadorSlot
    bool operadorSlot(std::string &strOperadorSlot);
    /// Valida si en la cadena continua un lowerKeyword
    bool lowerKeyword(std::string &strLowerKeyword);
    /// Valida si en la cadena continua un capKeyword
    bool capKeyword(std::string &strCapKeyword);

    /// Saltea los espacios
    void skipSpaces();
    /// Valida si el string a continuaci n de la cadena es el buscado.
    bool isString(const std::string strMatch);
    /// Valida si en la cadena continua una letra minuscula
    bool isLowercaseLetter();
    /// Valida si en la cadena continua una letra mayuscula
    bool isUppercaseLetter();
    /// Valida si en la cadena continua una letra
    bool isLetter();
```

```
    /// Valida si en la cadena continua un signo + -
    bool isSign();
    /// Valida si en la cadena continua un digito
    bool isDigit();
    /// Valida si en la cadena continua un digito o una letra
    bool isAlpha();

    //Devuelven los objetos, ya sea creandolos
    //o instanciandolos.
    /** Valida si en la cadena continua un nil obj y si es asi le pide a la
     * m quina virtual que lo cree.
     *
     */
    Object* nilObj();
    /** Valida si en la cadena continua un bool obj y si es asi le pide a la
     * m quina virtual que lo cree.
     *
     */
    Object* boolObj();
    /** Valida si en la cadena continua un string obj y si es asi le pide a la
     * m quina virtual que lo cree.
     *
     */
    Object* stringObj();
    /** Valida si en la cadena continua un number obj y si es asi le pide a la
     * m quina virtual que lo cree.
     *
     */
    Object* numberObj();
    /** Valida si en la cadena continua un objeto y si es asi le pide a la
     * m quina virtual que cree un objeto vac o al c al luego se le cargar n
     * los slots.
     *
     */
    Object* objectObj();
    /** Valida si en la cadena continua un name y si es asi se pide un lookup
     * de ese objeto para recuperarlo y devolverlo.
     *
     */
    Object* nameObj(Object* &context);

    /// Valida si en la cadena continua un nil
    bool nil();
    /// Valida si en la cadena continua un true
    bool isTrue();
    /// Valida si en la cadena continua un false
    bool isFalse();
    /// Valida si en la cadena continua un name
    bool name(std::string &strName);
    /// Valida si en la cadena continua un string
    std::string string();
    /// Valida si en la cadena continua un number
    bool number(float &number);
};

#endif /* SERVER_PARSER_H_ */
```

28 server_parser.cpp

```
#include "server_parser.h"

#include <vector>
#include <tuple>
#include <regex>
#include <sstream>
#include <string>
#include <iostream>
#include "server_object.h"
#include "common_define.h"

Parser::Parser(VirtualMachine &vm, Object* context) :
        vm(vm) {
    this->pCad = 0;
    this->flagExecute = 0;
    this->context = context;
}

std::vector<Object*> Parser::parse(std::string &cad) {
    //Si se utiliza el caracter especial de protocolo se lanza excepcion
    if (cad.find(CHAR_SEPARADOR) != std::string::npos) {
        std::stringstream ss;
        std::string s;
        char c = CHAR_SEPARADOR;
        ss << c;
        ss >> s;
        std::string msg = "El␣caracter␣especial␣" + s
                + "␣est␣prohibido␣por␣protocolo.";
        throw std::runtime_error(msg);
    }

    this->cad = cad;
    std::vector<Object*> objects = script();
    return objects;
}

std::vector<Object*> Parser::script() {
    if (debug)
        std::cout << "script␣pos:␣" << pCad << std::endl;
    flagExecute++;
    //int _pCad = pCad; //checkpoint
    Object* obj = nullptr;
    std::vector<Object*> objects;

    int pLastExpression = pCad;
    while (pCad < cad.size()) {
        if (((obj = expression()) != nullptr) and isString(PUNTO)) {
            objects.push_back(obj);
            pLastExpression = pCad;
        } else {
            pCad = pLastExpression;
            //destruir objeto creado y vaciar el vector
            break;
        }
    }
```

```
    flagExecute--;
    return objects;
}

Object * Parser::expression() {
    if (debug)
        std::cout << "expression␣pos:␣" << pCad << std::endl;

    int _pCad = pCad; //checkpoint
    Object* obj;

    if ((obj = keywordMessage()) != nullptr)
        return obj;
    else if ((obj = binaryMessage()) != nullptr)
        return obj;
    else if ((obj = unaryMessage()) != nullptr)
        return obj;
    else if ((obj = expressionCP()) != nullptr)
        return obj;

    pCad = _pCad;
    return obj;
}

Object * Parser::expressionCP() {
    if (debug)
        std::cout << "expressionCP␣pos:␣" << pCad << std::endl;

    int _pCad = pCad; //checkpoint
    Object* obj;

    if ((obj = expressionP()) != nullptr)
        return obj;
    else if ((obj = constant()) != nullptr)
        return obj;

    pCad = _pCad;
    return obj;
}

Object * Parser::expressionP() {
    if (debug)
        std::cout << "expressionP␣pos:␣" << pCad << std::endl;

    int _pCad = pCad; //checkpoint
    Object* obj = nullptr;

    if (isString(P_LEFT) and ((obj = expression()) != nullptr)
            and isString(P_RIGHT))
        return obj;

    pCad = _pCad;
    return obj;
}

Object * Parser::keywordMessage() {
    //Los keywordMessage solo soportan hasta un argumento.
```

```
    if (debug)
        std::cout << "keywordMessage pos: " << pCad << std::endl;

    int _pCad = pCad; //checkpoint
    Object* obj;
    Object* arg;
    std::string strLowerKeyword;

    if ((obj = receiver()) != nullptr and lowerKeyword(strLowerKeyword)
            and isString(OP_ARG) and (arg = expressionCP()) != nullptr) {
        std::vector<Object*> args = { arg };
        obj = receiveMessage(obj, strLowerKeyword, args);
        if (obj != nullptr)
            return obj;
    }
    pCad = _pCad;

    if (lowerKeyword(strLowerKeyword) and isString(OP_ARG) and (arg =
            expressionCP()) != nullptr) {
        std::vector<Object*> args = { arg };
        obj = receiveMessage(context, strLowerKeyword, args);
        if (obj != nullptr)
            return obj;
    }
    pCad = _pCad;

    return nullptr;
}

Object * Parser::binaryMessage() {
    if (debug)
        std::cout << "binaryMessage pos: " << pCad << std::endl;

    int _pCad = pCad; //checkpoint
    Object* obj;
    Object* arg;
    std::string strOp;

    if ((obj = receiver()) != nullptr and operador(strOp) and (arg =
            expressionCP()) != nullptr) {
        std::vector<Object*> args = { arg };
        obj = receiveMessage(obj, strOp, args);
        if (obj != nullptr)
            return obj;
    }

    pCad = _pCad;
    return nullptr;
}

Object * Parser::unaryMessage() {
    if (debug)
        std::cout << "unaryMessage pos: " << pCad << std::endl;

    int _pCad = pCad; //checkpoint
    Object* obj;
    std::string strName;
```

```
    //Probamos "receiver name"
    if ((obj = receiver()) != nullptr and name(strName)) {
        std::vector<Object*> args = { };
        obj = receiveMessage(obj, strName, args);
        if (obj != nullptr)
            return obj;
    }
    pCad = _pCad;

    //Probamos "receiver" (instancia el objeto y lo devuelve)
    if ((obj = receiver()) != nullptr) {
        return obj;
    }

    pCad = _pCad;
    return nullptr;
}

Object * Parser::receiveMessage(Object* obj, std::string &strName,
        std::vector<Object*> &args) {
    Object* objMessage;
    //Solo ejecutamos si es el script principal y no uno anidado.
    if (flagExecute == 1) {
        if (!obj->isDataObject(strName)) {
            //El objeto mensaje es un method object

            //Algunos m todos nativos se clonan y otros no
            //Pero todos los m todos nativos son considerados method objects.
            if (strName == ADD_SLOTS_METHOD || strName == REMOVE_SLOTS_METHOD
                    || strName == PRINTOBJ_METHOD || strName == PRINT_METHOD
                    || strName == CLONE_METHOD || strName == COLLECT_METHOD) {
                objMessage = obj->recvMessage(strName, args, false);
            } else {
                objMessage = obj->recvMessage(strName, args, true);
            }

            std::string code = objMessage->getCodeSegment();
            if (objMessage->isDataObject())
                // El objMessage obtenido se devuelve si es un data object.
                obj = objMessage;
            else {
                // El objMessage obtenido se ejecuta si no es un data object.
                Parser unParser(vm, objMessage);
                std::string self = SELF;
                objMessage->addSlot(self, obj, true, true, false);
                std::vector<Object*> _vector = unParser.parse(code);
                objMessage->removeSlot(self);
                obj = _vector[_vector.size() - 1];
            }
        } else {
            //El mensaje no ser  clonado por ser un data object.
            //En cambio se retornar  el data object.
            obj = obj->recvMessage(strName, args, false);
        }
    }
    return obj;
```

```
}

Object * Parser::receiver() {
    if (debug)
        std::cout << "receiver pos: " << pCad << std::endl;

    int _pCad = pCad; //checkpoint;
    Object* obj;
    if ((obj = expressionCP()) != nullptr)
        return obj;

    pCad = _pCad;
    return obj;
}

bool Parser::isLowercaseLetter() {
    int _pCad = pCad; //checkpoint;
    char cCad = cad[pCad];
    if ('a' <= cCad and cCad <= 'z') {
        pCad++;
        return true;
    }
    pCad = _pCad;
    return false;
}

bool Parser::isUppercaseLetter() {
    int _pCad = pCad; //checkpoint;
    char cCad = cad[pCad];
    if ('A' <= cCad and cCad <= 'Z') {
        pCad++;
        return true;
    }
    pCad = _pCad;
    return false;
}

bool Parser::isLetter() {
    int _pCad = pCad; //checkpoint;
    if (isLowercaseLetter() or isUppercaseLetter())
        return true;
    pCad = _pCad;
    return false;
}

bool Parser::isSign() {
    int _pCad = pCad; //checkpoint;
    char cCad = cad[pCad];
    if (cCad == '-' or cCad == '+') {
        pCad++;
        return true;
    }
    pCad = _pCad;
    return false;
}

bool Parser::isDigit() {
```

```
    int _pCad = pCad; //checkpoint;
    char cCad = cad[pCad];
    if ('0' <= cCad and cCad <= '9') {
        pCad++;
        return true;
    }
    pCad = _pCad;
    return false;
}

bool Parser::isAlpha() {
    int _pCad = pCad; //checkpoint;
    if (isDigit() or isLetter())
        return true;
    pCad = _pCad;
    return false;
}

bool Parser::name(std::string &strName) {
    if (debug)
        std::cout << "name␣pos:␣" << pCad << std::endl;

    int _pCad = pCad; //checkpoint
    skipSpaces();
    strName = "";
    char cCad = cad[pCad];
    if (isLowercaseLetter()) {
        strName += cCad;
        while (pCad < cad.size()) {
            cCad = cad[pCad];
            if (isAlpha())
                strName += cCad;
            else
                break;
        }
    }

    if (strName == "") {
        pCad = _pCad; //checkpoint
        return false;
    } else
        return true;
}

std::string Parser::string() {
    //No soporta strings con comillas simples escapeadas por comillas dobles.
    if (debug)
        std::cout << "string␣pos:␣" << pCad << std::endl;

    int _pCad = pCad; //checkpoint
    skipSpaces();
    std::string strString = "";
    bool esString = false;
    char cCad = cad[pCad];
    if (isString("'")) {
        strString += cCad;
        while (pCad < cad.size() and !esString) {
```

```
            cCad = cad[pCad];
            strString += cCad;
            pCad++;
            if ('\'' == cCad)
                esString = true;
        }
    }

    if (!esString) {
        pCad = _pCad;
        strString = "";
    }
    return strString;
}

bool Parser::number(float &number) {
    //No soporta n meros que no sean enteros.
    if (debug)
        std::cout << "number pos: " << pCad << std::endl;

    skipSpaces();
    std::string strNumber = "";
    char cCad = cad[pCad];
    if (isSign() or isDigit()) {
        strNumber += cCad;
        while (pCad < cad.size()) {
            cCad = cad[pCad];
            if (isDigit())
                strNumber += cCad;
            else
                break;
        }
    }

    if (strNumber != "") {
        number = std::stof(strNumber);
        return true;
    } else
        return false;
}

bool Parser::lowerKeyword(std::string &strLowerKeyword) {
    if (debug)
        std::cout << "lowerKeyword pos: " << pCad << std::endl;

    int _pCad = pCad; //checkpoint
    skipSpaces();
    strLowerKeyword = "";
    char cCad = cad[pCad];
    if (isLowercaseLetter() or isString("_")) {
        strLowerKeyword += cCad;
        while (pCad < cad.size()) {
            cCad = cad[pCad];
            if (isAlpha())
                strLowerKeyword += cCad;
            else
                break;
```

```
        }
    }

    if (strLowerKeyword == "") {
        pCad = _pCad;
        return false;
    } else
        return true;
}

bool Parser::capKeyword(std::string &strCapKeyword) {
    if (debug)
        std::cout << "capKeyword␣pos:␣" << pCad << std::endl;

    int _pCad = pCad; //checkpoint
    skipSpaces();
    strCapKeyword = "";
    char cCad = cad[pCad];
    if (isUppercaseLetter() or isString("_")) {
        strCapKeyword += cCad;
        while (pCad < cad.size()) {
            cCad = cad[pCad];
            if (isAlpha()) {
                strCapKeyword += cCad;
            } else
                break;
        }
    }

    if (strCapKeyword == "") {
        pCad = _pCad;
        return false;
    } else
        return true;
}

Object * Parser::objectObj() {
    if (debug)
        std::cout << "object␣pos:␣" << pCad << std::endl;

    int _pCad = pCad; //checkpoint
    Object* obj;
    obj = vm.createEmptyObject();
    int inicioScript, finScript;

    if (isString(P_LEFT) and isString(SLOT_LIST_SEP) and slotList(obj)
            and isString(SLOT_LIST_SEP) and (inicioScript = pCad)
            and (script().size() >= 0) and (finScript = pCad)
            and isString(P_RIGHT)) {
        obj->setCodeSegment(cad.substr(inicioScript, finScript - inicioScript));
        return obj;
    } else {
        pCad = _pCad;
        return nullptr;
    }
    return obj;
}
```

```
bool Parser::slotList(Object* objContenedor) {
    if (debug)
        std::cout << "slotList pos: " << pCad << std::endl;
    int tipoSlot; //0 normal, 1 argumento, 2 parent
    std::string strName;
    std::string strOpSlot;
    Object* objSlot;

    int pLastSlot = pCad;
    while (pCad < cad.size()) {
        if (slotNameExtended(tipoSlot, strName) and (operadorSlot(strOpSlot))
                and (objSlot = expression()) and isString(PUNTO)) {
            bool esMutable = true;
            bool esParent = false;
            bool esArgument = false;
            if (tipoSlot == 1)
                esArgument = true;
            else if (tipoSlot == 2)
                esParent = true;

            if (strOpSlot == OP_SLOT_INMUTABLE)
                esMutable = false;
            else if (strOpSlot == OP_SLOT_MUTABLE)
                esMutable = true;

            //Le insertamos al objeto contenedor el slot capturado.
            objContenedor->addSlot(strName, objSlot, esMutable, esParent,
                    esArgument);
            pLastSlot = pCad;
        } else {
            pCad = pLastSlot;
            if (slotNameExtended(tipoSlot, strName) and isString(PUNTO)) {
                objSlot = vm.createNil();
                bool esMutable = true;
                bool esParent = false;
                bool esArgument = false;
                if (tipoSlot == 1)
                    esArgument = true;
                else if (tipoSlot == 2)
                    esParent = true;

                //Le insertamos al objeto contenedor el slot capturado.
                objContenedor->addSlot(strName, objSlot, esMutable, esParent,
                        esArgument);
                pLastSlot = pCad;
            } else {
                pCad = pLastSlot;
                break;
            }
        }
    }
    //Si no se encontro una sintaxis v lida se da por supuesto
    //que el slotlist est  vacio, lo cual es una opci n v lida
    return true;
}
```

```
bool Parser::slotNameExtended(int &tipoSlot, std::string &strName) {
    if (debug)
        std::cout << "slotNameExtended␣pos:␣" << pCad << std::endl;

    skipSpaces();
    int _pCad = pCad; //checkpoint
    if (isString(OP_ARG) and name(strName)) {
        tipoSlot = 1;
    } else {
        pCad = _pCad;
        //Se agrega el soporte de operadores en los nombres de slots
        //que por enunciado no estaban soportados
        if (name(strName) and isString(OP_PARENT)) {
            tipoSlot = 2;
        } else {
            pCad = _pCad;
            if (name(strName) or operador(strName) or lowerKeyword(strName)) {
                tipoSlot = 0;
            }
        }
    }
    return true;
}

Object * Parser::constant() {
    if (debug)
        std::cout << "constant␣pos:␣" << pCad << std::endl;

    int _pCad = pCad; //checkpoint
    Object* obj;

    if ((obj = nilObj()) != nullptr)
        return obj;
    else if ((obj = boolObj()) != nullptr)
        return obj;
    else if ((obj = stringObj()) != nullptr)
        return obj;
    else if ((obj = numberObj()) != nullptr)
        return obj;
    else if ((obj = objectObj()) != nullptr)
        return obj;
    else if ((obj = nameObj(context)) != nullptr)
        return obj;
    else {
        pCad = _pCad;
        return nullptr;
    }
}

bool Parser::operador(std::string &strOperador) {
    if (debug)
        std::cout << "operador␣pos:␣" << pCad << std::endl;

    int _pCad = pCad; //checkpoint
    skipSpaces();

    if (isString(OP_SUMA)) {
```

```
        strOperador = OP_SUMA;
        return true;
    } else if (isString(OP_RESTA)) {
        strOperador = OP_RESTA;
        return true;
    } else if (isString(OP_MULTIPLICACION)) {
        strOperador = OP_MULTIPLICACION;
        return true;
    } else if (isString(OP_DIVISION)) {
        strOperador = OP_DIVISION;
        return true;
    } else if (isString(OP_DISTINTO)) {
        strOperador = OP_DISTINTO;
        return true;
    } else if (isString(OP_IGUAL)) {
        strOperador = OP_IGUAL;
        return true;
    } else {
        strOperador = "";
        pCad = _pCad;
        return false;
    }
}

bool Parser::operadorSlot(std::string &strOperadorSlot) {
    if (debug)
        std::cout << "operadorSlot␣pos:␣" << pCad << std::endl;

    int _pCad = pCad; //checkpoint
    skipSpaces();

    if (isString(OP_SLOT_INMUTABLE)) {
        strOperadorSlot = OP_SLOT_INMUTABLE;
        return true;
    } else if (isString(OP_SLOT_MUTABLE)) {
        strOperadorSlot = OP_SLOT_MUTABLE;
        return true;
    } else {
        strOperadorSlot = "";
        pCad = _pCad;
        return false;
    }
}

void Parser::skipSpaces() {
    //if (debug)
    //  std::cout << "skipSpaces pos: " << pCad << std::endl;

    char cCad;
    bool salir = false;
    while (pCad < cad.size() and !salir) {
        cCad = cad[pCad];
        if (cCad == '␣' or cCad == '\t' or cCad == '\n')
            pCad++;
        else
            salir = true;
    }
```

```
}

bool Parser::isString(const std::string strMatch) {
    if (debug)
        std::cout << strMatch << "␣pos:␣" << pCad << std::endl;

    int _pCad = pCad; //checkpoint
    skipSpaces();
    bool isMatch = true;

    for (uint32_t i = 0; i < strMatch.size(); i++) {
        if ((pCad < cad.size()) and (strMatch[i] == cad[pCad]))
            pCad++;
        else
            isMatch = false;
    }

    if (!isMatch)
        pCad = _pCad;
    return isMatch;
}

Object * Parser::nilObj() {
    int _pCad = pCad; //checkpoint
    Object *obj = nullptr;
    skipSpaces();

    if (isString(NIL)) {
        obj = vm.createNil();
        return obj;
    }

    pCad = _pCad;
    return obj;
}

Object * Parser::boolObj() {
    int _pCad = pCad; //checkpoint
    Object *obj = nullptr;
    skipSpaces();

    if (isString(TRUE_STR)) {
        obj = vm.createBoolean(true);
        return obj;
    } else if (isString(FALSE_STR)) {
        obj = vm.createBoolean(false);
        return obj;
    }

    pCad = _pCad;
    return obj;
}

Object * Parser::stringObj() {
    int _pCad = pCad; //checkpoint
    Object *obj = nullptr;
    skipSpaces();
```

```
    std::string strString = string();

    if (strString != "") {
        obj = vm.createString(strString);
        return obj;
    }

    pCad = _pCad;
    return obj;
}

Object * Parser::numberObj() {
    int _pCad = pCad; //checkpoint
    Object *obj = nullptr;
    skipSpaces();

    float tempNumber = 0;
    if (number(tempNumber)) {
        obj = vm.createNumber(tempNumber);
        return obj;
    }

    pCad = _pCad;
    return obj;
}

Object * Parser::nameObj(Object* &context) {
    int _pCad = pCad; //checkpoint
    Object *obj = nullptr;
    skipSpaces();
    std::string strName;

    if (name(strName)) {
        if (flagExecute == 1) {
            std::vector<Object*> args = { };
            obj = receiveMessage(context, strName, args);
        } else {
            //Se utiliza cuando no se debe ejecutar el codigo
            //Pero se debe devolver algo v lido.
            obj = vm.createNil();
        }
    } else
        pCad = _pCad;

    return obj;
}
```

29 server_accepter.h

```
#ifndef SERVER_ACCEPTER_H_
#define SERVER_ACCEPTER_H_

#include <stdexcept>
#include <vector>
#include "common_socket.h"
#include "server_proxy_client.h"
#include "common_thread.h"
#include "server_server.h"
#include "server_workspace.h"

/** Es el encargado de aceptar nuevos clientes abriendo
 * proxys en nuevos hilos. Un hilo por cada cliente conectado.
 *
 */
class Accepter: public Thread {
private:
    std::vector<ProxyClient*> programThreads;
    Server &server;
    Socket socket;
    bool interrupt_task;

public:
    /// Constructor
    /**
     * @param port puerto donde escuchar
     * @param server referencia al Server
     * @param workspace  puntero a Workspace.
     */
    Accepter(uint32_t port, Server &server);

    /** Constructor por copia deshabilitado
     *
     */
    Accepter(const Accepter&) = delete;

    /** Constructor por moviemiento deshabilitado
     *
     */
    Accepter(Accepter&&) = delete;

    /** Operador asignacion deshabilitado
     *
     */
    Accepter& operator=(const Accepter&) = delete;

    /** Operador asignacion por moviemiento deshabilitado
     *
     */
    Accepter& operator=(Accepter&&) = delete;

    /// Destructor
    ~Accepter();

    /** Metodo que sirve para la interrupcion del proceso.
```

```
     *
     */
    void interrupt();

    /** Metodo principal de la clase. Hace los procesamientos
     *
     */
    virtual void run();

private:
    /** Revisa si los clientes que tiene conectados terminaron su ejecucion,
     * luego limpia los recursos utilizados
     *
     */
    void collectClosedClients();
};

#endif /* SERVER_ACCEPTER_H_ */
```

30 server_accepter.cpp

```
#include "server_accepter.h"
#include <stdexcept>
#include <vector>

Accepter::Accepter(uint32_t port, Server &server) :
        programThreads(), server(server), socket(port) {
    interrupt_task = false;
    socket.bind_and_listen();
}

Accepter::~Accepter() {
    socket.shutdown();
}

void Accepter::run() {
    try {
        while (!interrupt_task) {
            Socket *sck = socket.accept();
            collectClosedClients();
            ProxyClient *newProxy = new ProxyClient(*sck, server);
            programThreads.push_back(newProxy);
            newProxy->start();
        }
    } catch (const std::runtime_error &e) {
        if (!interrupt_task)
            throw e;
    }
}

void Accepter::interrupt() {
    this->interrupt_task = true;
    this->socket.shutdown();
    for (uint32_t i = 0; i < programThreads.size(); i++) {
        programThreads[i]->interrupt();
        programThreads[i]->join();
        delete programThreads[i];
    }
}

void Accepter::collectClosedClients() {
    std::vector<ProxyClient*> new_programThreads;

    for (uint32_t i = 0; i < programThreads.size(); i++) {
        if (programThreads[i]->is_finished()) {
            programThreads[i]->join();
            delete programThreads[i];
        } else {
            new_programThreads.push_back(programThreads[i]);
        }
    }

    new_programThreads.swap(programThreads);
    new_programThreads.clear();
}
```

31 server_server.h

```
#ifndef SERVER_SERVER_H_
#define SERVER_SERVER_H_

#include <map>
#include <tuple>
#include <stack>
#include <mutex>
#include <string>
#include <stdexcept>
#include <vector>
#include "server_object.h"
#include "common_thread.h"
#include "server_workspace.h"
#include "common_define.h"
#include "server_parser_protocolo_servidor.h"

/** Representa el modelo de negocio. Resuelve las peticiones de los
 * ProxyClient's y administra los recursos que se deben proteger.
 *
 */
class Server {
private:
    std::mutex m;
    typedef std::tuple<Workspace*, uint32_t> workspace_tuple;
    /** Mapa con clave ID workspaces y valor una tupla con
     * un puntero al workspace y la cantidad de clientes activos en el mismo.
     *
     */
    std::map<std::string, workspace_tuple> workspaces;

private:
    /** Retorna el workspaces en funci n del id solicitado
     * @param idWk id del workspaces
     */
    Workspace* getWorkspace(const std::string &idWk);

public:
    /** Constructor
     */
    Server();
    /** Destructor
     */
    ~Server();

    Server(const Server&) = delete;
    Server(Server&&) = delete;
    Server& operator=(const Server&) = delete;
    Server& operator=(Server&&) = delete;

    /** Acumula en el contador de clientes del workspace en el mapa workspaces
     * @param name id del workspace al que se conecta el cliente
     */
    void loadWorkspace(std::string name);
    /** Retorna una cadena por formato de protocolo con la lista de
     * workspaces dispponibles.
```

```
 */
std::vector<std::string> availableWorkspace();
/** Inicializa un nuevo workspace con el nombre pasado por parametro.
 * @param name id del nuevo workspace
 */
void newWorkspace(std::string name);
/** Desacumula en el contador de clientes del workspace en el mapa
    workspaces
 * @param name id del workspace del que se desconecta el cliente
 */
void closeWorkspace(std::string name);
/** Elimina el workspace con el nombre pasado por parametro.
 * @param name id del workspace a eliminar
 */
void deleteWorkspace(std::string name);
/** Le pide al workspace que ejecute codigo self con el contexto dado.
 * @param idWk id del workspace
 * @param idObj id del objeto de ese workspace
 * @param code script de c digo self
 */
std::string receiveCode(const std::string &idWk, uint32_t &idObj,
        std::string &code);
/** Le pide el lobby al workspace y genera la cadena del objeto por
    protocolo
 * @param idWk id del workspace
 * @param idObj id del objeto de ese workspace
 */
std::string getLobby(const std::string &idWk, uint32_t &idObj);
/** Le pide el objeto al workspace y genera la cadena del objeto por
    protocolo
 * @param idWk id del workspace
 * @param idObj id del objeto de ese workspace
 */
std::string getObj(const std::string &idWk, uint32_t &idObj);
/** Setea el nombre del objeto y genera la cadena del objeto por protocolo
 * @param idWk id del workspace
 * @param idObj id del objeto de ese workspace
 * @param cad nuevo nombre del objeto
 */
std::string setObjName(const std::string &idWk, uint32_t &idObj,
        const std::string &cad);
/** Setea el bloque de c digo del objeto y genera la cadena del objeto por
 *  protocolo
 * @param idWk id del workspace
 * @param idObj id del objeto de ese workspace
 * @param cad nuevo bloque de c digo
 */
std::string setCodeSegment(const std::string &idWk, uint32_t &idObj,
        const std::string &cad);
/** Genera la cadena por protocolo del objeto contenido en el slot
 * @param idWk id del workspace
 * @param idObj id del objeto de ese workspace
 * @param cad nombre del slot
 */
std::string getSlotObj(const std::string &idWk, uint32_t &idObj,
        const std::string &cad);
/** Cambia la mutabilidad del slot y genera la cadena por protocolo del
```

```
         objeto
      * @param idWk id del workspace
      * @param idObj id del objeto de ese workspace
      * @param cad nombre del slot
      */
     std::string swapMutability(const std::string &idWk, uint32_t &idObj,
             const std::string &cad);
};

#endif /* SERVER_SERVER_H_ */
```

32 server_server.cpp

```
#include "server_server.h"

Server::Server() {
}

Server::~Server() {
    std::stack<Workspace*> stack;

    for (auto it = workspaces.begin(); it != workspaces.end(); ++it) {
        stack.push(std::get<0>(it->second));
    }

    while (stack.size() > 0) {
        auto wk = stack.top();
        stack.pop();
        delete wk;
    }
}

void Server::loadWorkspace(std::string name) {
    m.lock();
    auto it = workspaces.find(name);
    if (it == workspaces.end()) {
        m.unlock();
        throw std::runtime_error("No existe el workspace");
    }

    workspace_tuple tuple = it->second;
    std::get<1>(tuple) += 1;
    it->second = tuple;
    m.unlock();
}

std::vector<std::string> Server::availableWorkspace() {
    std::vector<std::string> _workspaces;
    m.lock();
    for (auto it = workspaces.begin(); it != workspaces.end(); ++it) {
        _workspaces.push_back(it->first);
    }
    m.unlock();
    return _workspaces;
}

void Server::newWorkspace(std::string name) {
    m.lock();
    auto it = workspaces.find(name);

    if (it == workspaces.end()) {
        Workspace *wk = new Workspace();
        workspace_tuple tuple = std::make_tuple(wk, 0);
        workspaces.insert(std::make_pair(name, tuple));
        m.unlock();
    } else {
        m.unlock();
        std::string error = "Ya existe un workspace llamado " + name + ".";
```

```
        throw std::runtime_error(error);
    }

}

void Server::closeWorkspace(std::string name) {
    m.lock();
    auto it = workspaces.find(name);
    if (it == workspaces.end()) {
        m.unlock();
        throw std::runtime_error("No existe el workspace");
    }

    workspace_tuple tuple = it->second;
    std::get<1>(tuple) -= 1;
    it->second = tuple;
    m.unlock();
}

void Server::deleteWorkspace(std::string name) {
    m.lock();
    auto it = workspaces.find(name);
    if (it == workspaces.end()) {
        m.unlock();
        throw std::runtime_error("No existe el workspace");
    } else {
        auto tuple = it->second;
        if (std::get<1>(tuple) == 0) {
            workspaces.erase(name);
            m.unlock();
        } else {
            m.unlock();
            std::string error = "El workspace " + name
                    + " tiene un cliente conectado.";
            throw std::runtime_error(error);
        }
    }

}

Workspace* Server::getWorkspace(const std::string &idWk) {
    auto it = workspaces.find(idWk);
    if (it == workspaces.end()) {
        std::string error = "El workspace " + idWk + " no existe";
        throw std::runtime_error(error);
    }
    Workspace* _wk = std::get<0>(it->second);
    return _wk;
}

std::string Server::receiveCode(const std::string &idWk, uint32_t &idObj,
        std::string &code) {
    std::string msg = "";
    try {
        m.lock();
        Workspace* wk = getWorkspace(idWk);
        Object *context = wk->findObjectById(idObj);
```

```
        uint32_t idRet;
        idRet = wk->receive(context, code);
        Object *objRet = wk->findObjectById(idRet);
        idObj = idRet;
        m.unlock();
        msg = ParserProtocoloServidor(objRet).getString();
    } catch (...) {
        m.unlock();
        throw;
    }
    return msg;
}

std::string Server::getLobby(const std::string &idWk, uint32_t &idObj) {
    idObj = ID_LOBBY;
    return getObj(idWk, idObj);
}

std::string Server::getObj(const std::string &idWk, uint32_t &idObj) {
    Workspace* wk;
    Object* objRet;
    try {
        wk = getWorkspace(idWk);
        objRet = wk->findObjectById(idObj);
    } catch (...) {
        m.unlock();
        throw;
    }
    return ParserProtocoloServidor(objRet).getString();
}

std::string Server::setObjName(const std::string &idWk, uint32_t &idObj,
        const std::string &cad) {
    Workspace* wk;
    Object* objRet;
    try {
        wk = getWorkspace(idWk);
        objRet = wk->findObjectById(idObj);
        objRet->setName(cad);
    } catch (...) {
        m.unlock();
        throw;
    }
    return ParserProtocoloServidor(objRet).getString();
}

std::string Server::setCodeSegment(const std::string &idWk, uint32_t &idObj,
        const std::string &cad) {
    Workspace* wk;
    Object* objRet;
    try {
        wk = getWorkspace(idWk);
        objRet = wk->findObjectById(idObj);
        objRet->setCodeSegment(cad);
    } catch (...) {
        m.unlock();
        throw;
```

```
    }
    return ParserProtocoloServidor(objRet).getString();
}

std::string Server::getSlotObj(const std::string &idWk, uint32_t &idObj,
        const std::string &cad) {
    Workspace* wk;
    Object* obj;
    m.lock();
    try {
        wk = getWorkspace(idWk);
        obj = wk->findObjectById(idObj);
    } catch (...) {
        m.unlock();
        throw;
    }
    std::string retVal;
    auto slots = obj->getSlots();
    auto it = slots.find(cad);
    if (it != slots.end()) {
        Object* objRet = std::get<0>(it->second);
        if (!objRet) {
            m.unlock();
            throw std::runtime_error("El␣slot␣tiene␣un␣puntero␣nulo");
        } else {
            idObj = objRet->getId();
            retVal = ParserProtocoloServidor(objRet).getString();
        }
    } else {
        m.unlock();
        throw std::runtime_error("El␣slot␣buscado␣no␣existe");
    }
    m.unlock();
    return retVal;
}

std::string Server::swapMutability(const std::string &idWk, uint32_t &idObj,
        const std::string &cad) {
    Workspace* wk;
    Object* obj;
    m.lock();
    try {
        wk = getWorkspace(idWk);
        obj = wk->findObjectById(idObj);
        obj->swapSlotMutability(cad);
    } catch (...) {
        m.unlock();
        throw;
    }
    idObj = obj->getId();
    m.unlock();
    return ParserProtocoloServidor(obj).getString();
}
```

33 server_proxy_client.h

```
#ifndef SERVER_PROXYCLIENT_H_
#define SERVER_PROXYCLIENT_H_

#include "common_proxy.h"

/** El ProxyClient es el encargado de responder las peticiones del
 * cliente (ProxyServer). Para resolver las peticiones delega las
 * consultas al servidor(Server/Modelo de Negocio).
 * El ProxyClient solo conoce los IDs del Workspace y de los objetos
 * con los que trabaja.
 *
 */
class ProxyClient: public Proxy {
private:
    /// Servidor al que el proxy le hace las consultas
    Server &server;
    /// Workspace en el que se encuentra el cliente
    std::string idWorkspace;
    /// Pila de IDs de objetos vistos por el cliente
    std::stack<uint32_t> seenObj;
    /// Socket asociado al proxyClient
    Socket* sckptr = nullptr;
public:
    /** Constructor.
     * @param socket socket sobre el cual trabajar
     * @param server referencia al modelo de negocio
     */
    ProxyClient(Socket &socket, Server &server);

    /** Constructor por copia eliminado
     *
     */
    ProxyClient(const ProxyClient&) = delete;

    /** Constructor por moviemiento eliminado
     *
     */
    ProxyClient(ProxyClient&&) = delete;

    /** Operador asignacion eliminado
     *
     */
    ProxyClient& operator=(const ProxyClient&) = delete;

    /** Operador asignacion por moviemiento eliminado
     *
     */
    ProxyClient& operator=(ProxyClient&&) = delete;

    /** Destructor.
     *
     */
    ~ProxyClient();

    /** Metodo que sirve para procesar la solicitud que le envia el cliente.
```

```
     *
     */
    virtual void run();
private:
    /** Le pide al servidor que ejecute un script de c digo self con el
     * entorno/contexto de lobby y le retorna al cliente la respuesta.
     * @param cad script a procesar por el servidor.
     */
    void execLobbyCMD(std::string &cad);
    /** Le pide al servidor que ejecute un script de c digo self con el
     * entorno/contexto del objeto que ve el cliente y le retorna al cliente
     *  la respuesta.
     * @param cad script a procesar por el servidor.
     */
    void execLocalCMD(std::string &cad);
    /** Le pide al servidor la cadena que representa por protocolo a lobby
     *  y le retorna al cliente la respuesta.
     */
    void showLobby();
    /** Le pide al servidor la cadena que representa por protocolo al
     * objeto que esta viendo el cliente para actualizar las novedades
     * y le retorna al cliente la respuesta.
     */
    void execRefresh();
    /** Le pide al servidor que le setee el nombre al objeto que ve el cliente.
     * Retorna al cliente la respuesta con el objeto modificado.
     * @param cad nuevo nombre.
     */
    void setObjName(const std::string &cad);
    /** Le pide al servidor que le setee el bloque de c digo al objeto que ve
     * el cliente y retorna al cliente la respuesta con el objeto modificado.
     * @param cad nuevo bloque de codigo.
     */
    void setCodeSegment(const std::string &cad);
    /** Le pide al servidor el objeto contenido en el slot del objeto que el
     * cliente est  viendo.
     * @param cad nombre del slot en el objeto que ve el cliente.
     */
    void getSlotObj(const std::string &cad);
    /** Le pide al servidor cambiar la mutabilidad del slot del objeto que el
     * cliente est  viendo.
     * @param cad nombre del slot en el objeto que ve el cliente.
     */
    void swapMutability(const std::string &cad);
    /** Le pide al servidor el objeto anterior de la pila seenObj
     * y se lo devuelve al cliente.
     */
    void goBack();
    /** Le pide al servidor una lista de workspaces existentes
     * y le devuelve esa lista al cliente.
     */
    void availableWks();
    /** Le indica al servidor que un cliente va a entrar a un workspace
     * por lo que el servidor le debe retornar el lobby de ese workspace
     * para devolverselo al cliente.
     * @param cad nombre del workspace a cargar.
     */
```

```
    void loadWks(const std::string &cad);
    /** Le indica al servidor que se debe crear un nuevo workspace
     * y le retorna al cliente el lobby que le devolvi el server.
     * @param cad nombre del nuevo workspace.
     */
    void newWks(const std::string &cad);
    /** Le indica al servidor que se debe eliminar un workspace
     * y le retorna al cliente la nueva lista de workspace disponibles.
     * @param cad nombre del workspacer a eliminar.
     */
    void deleteWks(const std::string &cad);
    /** Le indica al servidor que el cliente se desconecta del workspace
     *  actual.
     */
    void closeWks();

    /** Devuelve el ID del objeto que est  mas arriba en la pila de seenObj.
     * Es decir de los objetos vistos por el cliente hasta el momento.
     */
    uint32_t topObj();
};

#endif /* SERVER_PROXYCLIENT_H_ */
```

34 server_proxy_client.cpp

```
#include "server_proxy_client.h"
#include <string>
#include <vector>

ProxyClient::ProxyClient(Socket &socket, Server &server) :
        Proxy(socket), server(server) {
    sckptr = &socket;
    seenObj.push(ID_LOBBY);
}

ProxyClient::~ProxyClient() {
    delete sckptr;
}

void ProxyClient::execLobbyCMD(std::string &cad) {
    try {
        std::string sendMsg;
        uint32_t lobbyId = ID_LOBBY;
        sendMsg = server.receiveCode(idWorkspace, lobbyId, cad);
        sendOK(sendMsg);
    } catch (const std::runtime_error &e) {
        sendError(e.what());
    } catch (...) {
        sendError("Error desconocido.");
    }
}

void ProxyClient::execLocalCMD(std::string &cad) {
    try {
        std::string sendMsg;
        uint32_t idObj = topObj();
        sendMsg = server.receiveCode(idWorkspace, idObj, cad);
        if (idObj != topObj())
            seenObj.push(idObj);
        sendOK(sendMsg);
    } catch (const std::runtime_error &e) {
        sendError(e.what());
    } catch (...) {
        sendError("Error desconocido.");
    }
}

void ProxyClient::showLobby() {
    try {
        std::string sendMsg;
        uint32_t idObj;
        sendMsg = server.getLobby(idWorkspace, idObj);
        if (idObj != topObj())
            seenObj.push(idObj);
        sendOK(sendMsg);
    } catch (const std::runtime_error &e) {
        sendError(e.what());
    } catch (...) {
        sendError("Error desconocido.");
    }
```

```
}

void ProxyClient::execRefresh() {
    try {
        std::string sendMsg;
        uint32_t idObj = topObj();
        sendMsg = server.getObj(idWorkspace, idObj);
        if (idObj != topObj())
            seenObj.push(idObj);
        sendOK(sendMsg);
    } catch (const std::runtime_error &e) {
        sendError(e.what());
    } catch (...) {
        sendError("Error␣desconocido.");
    }
}

void ProxyClient::setObjName(const std::string &cad) {
    try {
        std::string sendMsg;
        uint32_t idObj = topObj();
        sendMsg = server.setObjName(idWorkspace, idObj, cad);
        if (idObj != topObj())
            seenObj.push(idObj);
        sendOK(sendMsg);
    } catch (const std::runtime_error &e) {
        sendError(e.what());
    } catch (...) {
        sendError("Error␣desconocido.");
    }
}

void ProxyClient::setCodeSegment(const std::string &cad) {
    try {
        std::string sendMsg;
        uint32_t idObj = topObj();
        sendMsg = server.setCodeSegment(idWorkspace, idObj, cad);
        if (idObj != topObj())
            seenObj.push(idObj);
        sendOK(sendMsg);
    } catch (const std::runtime_error &e) {
        sendError(e.what());
    } catch (...) {
        sendError("Error␣desconocido.");
    }
}

void ProxyClient::getSlotObj(const std::string &cad) {
    try {
        std::string sendMsg;
        uint32_t idObj = topObj();
        sendMsg = server.getSlotObj(idWorkspace, idObj, cad);
        if (idObj != topObj())
            seenObj.push(idObj);
        sendOK(sendMsg);
    } catch (const std::runtime_error &e) {
        sendError(e.what());
```

```
    } catch (...) {
        sendError("Error␣desconocido.");
    }
}

void ProxyClient::swapMutability(const std::string &cad) {
    try {
        std::string sendMsg;
        uint32_t idObj = topObj();
        sendMsg = server.swapMutability(idWorkspace, idObj, cad);
        if (idObj != topObj())
            seenObj.push(idObj);
        sendOK(sendMsg);
    } catch (const std::runtime_error &e) {
        sendError(e.what());
    } catch (...) {
        sendError("Error␣desconocido.");
    }
}

void ProxyClient::goBack() {
    try {
        std::string sendMsg;
        if (seenObj.size() > 1)
            seenObj.pop();

        uint32_t idObj = topObj();
        sendMsg = server.getObj(idWorkspace, idObj);
        if (idObj != topObj())
            seenObj.push(idObj);
        sendOK(sendMsg);
    } catch (const std::runtime_error &e) {
        sendError(e.what());
    } catch (...) {
        sendError("Error␣desconocido.");
    }
}

void ProxyClient::availableWks() {
    try {
        std::vector<std::string> vecWks = server.availableWorkspace();
        std::string msg;
        for (std::string str : vecWks) {
            msg += str + CHAR_SEPARADOR;
        }
        msg = msg.substr(0, msg.size() - 1);
        sendOKWks(msg);
    } catch (const std::runtime_error &e) {
        sendError(e.what());
    } catch (...) {
        sendError("Error␣desconocido.");
    }
}

void ProxyClient::loadWks(const std::string &cad) {
    try {
        idWorkspace = cad;
```

```
        server.loadWorkspace(cad);
        showLobby();
    } catch (const std::runtime_error &e) {
        idWorkspace.clear();
        sendError(e.what());
    } catch (...) {
        idWorkspace.clear();
        sendError("Error desconocido.");
    }
}

void ProxyClient::newWks(const std::string &cad) {
    try {
        server.newWorkspace(cad);
        availableWks();
    } catch (const std::runtime_error &e) {
        sendError(e.what());
    } catch (...) {
        sendError("Error desconocido.");
    }
}

void ProxyClient::deleteWks(const std::string &cad) {
    try {
        server.deleteWorkspace(cad);
        availableWks();
    } catch (const std::runtime_error &e) {
        sendError(e.what());
    } catch (...) {
        sendError("Error desconocido.");
    }
}

void ProxyClient::closeWks() {
    try {
        server.closeWorkspace(idWorkspace);
        availableWks();
        idWorkspace.clear();
    } catch (const std::runtime_error &e) {
        sendError(e.what());
    } catch (...) {
        sendError("Error desconocido.");
    }
}

void ProxyClient::run() {
    while (!_interrupt) {
        try {
            int s = this->receive();
            if (s == 0) {
                finished = true;
                break;
            }

            std::string cad = "";
            switch (this->message.getCommand()) {
            case EXEC_LOBBY_CMD: {
```

```
        cad = message.getMessage();
        execLobbyCMD(cad);
        break;
    }
    case SHOW_LOBBY: {
        showLobby();
        break;
    }
    case EXEC_REFRESH: {
        execRefresh();
        break;
    }
    case EXEC_LOCAL_CMD: {
        cad = message.getMessage();
        execLocalCMD(cad);
        break;
    }
    case ADD_SLOT: {
        cad += ADD_SLOTS_METHOD + OP_ARG + P_LEFT + SLOT_LIST_SEP;
        cad += message.getMessage();
        cad += SLOT_LIST_SEP + P_RIGHT + PUNTO;
        execLocalCMD(cad);
        break;
    }
    case REMOVE_SLOT: {
        cad += REMOVE_SLOTS_METHOD + OP_ARG + P_LEFT + SLOT_LIST_SEP;
        cad += message.getMessage();
        cad += PUNTO + SLOT_LIST_SEP + P_RIGHT + PUNTO;
        execLocalCMD(cad);
        break;
    }
    case SET_OBJ_NAME: {
        cad = message.getMessage();
        setObjName(cad);
        break;
    }
    case SET_CODESEGMENT: {
        cad = message.getMessage();
        setCodeSegment(cad);
        break;
    }
    case GET_SLOT_OBJ: {
        cad = message.getMessage();
        getSlotObj(cad);
        break;
    }
    case SWAP_MUTABILITY: {
        cad = message.getMessage();
        swapMutability(cad);
        break;
    }
    case GO_BACK: {
        cad = message.getMessage();
        goBack();
        break;
    }
    case AVAILABLE_WKS: {
```

```
                availableWks();
                break;
            }
            case LOAD_WK: {
                cad = message.getMessage();
                loadWks(cad);
                break;
            }
            case NEW_WK: {
                cad = message.getMessage();
                newWks(cad);
                break;
            }
            case DELETE_WK: {
                cad = message.getMessage();
                deleteWks(cad);
                break;
            }
            case CLOSE_WK: {
                cad = message.getMessage();
                closeWks();
                break;
            }
            default:
                sendError("Comando␣desconocido.");
            }
        } catch (const std::runtime_error &e) {
            if (!_interrupt)
                throw e;
        }
    }
}

uint32_t ProxyClient::topObj() {
    if (seenObj.size() > 0)
        return seenObj.top();

    throw std::runtime_error("Todavia␣no␣se␣visito␣ningun␣objeto.");
}
```

35 server_object.h

```
#ifndef COMMON_OBJECT_H_
#define COMMON_OBJECT_H_

#include <string>
#include <stdexcept>
#include <map>
#include <tuple>
#include <vector>
#include <stack>

/** Representa un objeto del lenguaje Self.
 *
 */
class Object {
public:
    /** Definicion del tipo delegate que es un puntero a funcion
     * que tiene la forma Object* funcion (const std::vector<Object*>&)
     */
    typedef Object* (Object::*delegate)(const std::vector<Object*>&);

    /** Formato que tiene el slot. \n
     * puntero a Object con la referencia al objeto  \n
     * booleano que indica si es mutable o no  \n
     * booleano que indica si el objeto apuntado es un parent slot  \n
     * booleano que indica si esta implementado nativamente  \n
     * booleano que indica si es un argumento (:)
     */
    typedef std::tuple<Object*, bool, bool, bool> slot_t;

    /** Formato que representa el diccionario interno.
     * Con string como clave que representa el nombre del
     * slot, y slot_t que representa el valor.
     */
    typedef std::map<std::string, slot_t> slot_map;

    /** Formato que representa las funciones nativas que estan
     * habilitadas.
     * En la primera parte de la tupla esta el puntero a funcion
     * En la segunda un booleano que indica si esta habilitado o no.
     */
    typedef std::tuple<delegate, bool> fpointTuple;

private:
    /// Representa los slots del objeto
    slot_map slots;

    /// Nombre del objeto
    std::string name;

    /// Representa el code segment del objeto
    std::string codeSegment;

    /** Diccionario con los metodos nativos. La clave es el nombre y el
     *  valor el puntero a la funcion.
     */
```

```
    std::map<std::string, fpointTuple> nativeMethods;

    /// Puntero al objeto lobby
    Object* lobby;

    /// id numerico (y unico) del objeto.
    uint32_t id = 0;

    /// Contador de objetos
    uint32_t idCounter = 1;

    /* Tupla con un puntero a objeto y un booleano indicando
     * si se recorrio o no (sirve para el algoritmo del GC)
     */
    typedef std::tuple<Object*, bool> tuple_createdObjects;

    /// Diccionario con todos los objetos creados.
    std::map<uint32_t, tuple_createdObjects> createdObjects;

    /// Indica si el objeto es de un tipo primitivo.
    bool isPrimitive = false;

private:
    /** Devuelve los parent slots que tiene el objeto.
     *
     */
    slot_map getParentSlots() const;

    /** Devuelve los parent slots del objeto pasado como argumento
     * @param pointer objeto sobre el cual devolver los parent slot.
     */
    Object::slot_map getParentSlots(Object* pointer) const;

    /** Este metodo sirve para buscar el objeto de un slot.
     * @param name nombre del slot a buscar
     * @param returnValue puntero a Object devuelto con el
     * puntero al slot.
     * @param function puntero a funcion devuelto si es un metodo
     * nativo.
     * \retval true si encontro el objeto seteando
     * el puntero.
     * \retval false, los punteros estan en nullptr.
     */
    bool findObject(std::string name, Object* &returnValue,
            delegate& function) const;

    /** Este metodo habilita los metodos nativos comunes a todos
     * los objetos. Estos son: _AddSlots, _RemoveSlots, clone y printObj.
     *
     */
    void configureNativeMethods();

    /** Recorre todos los slots del objeto y va marcando que es accesible
     *
     */
    void collect_internal();
```

```
/** Funcion nativa para agregar un slot.
 * @param args vector con Object* con los objetos
 * a agregar al slot.
 */
Object* _AddSlots(const std::vector<Object*>& args);

/** Funcion nativa para borrar un slot.
 * @param args vector con Object* con los objetos para borrar.
 */
Object* _RemoveSlots(const std::vector<Object*>& args);

/** Metodo nativo para clonar el objeto.
 * @param args vector de Object* vacio.
 */
Object* clone(const std::vector<Object*>& args);

// funciones Nativas
/** Metodo nativo para imprimir por pantalla.
 * @param args vector de Object* vacio.
 */
Object* print(const std::vector<Object*>& args);

/** Metodo nativo para imprimir por pantalla datos de debugging.
 * @param args vector de Object* vacio.
 */
Object* printObj(const std::vector<Object*>& args);

/** Metodo nativo para efectuar la multiplicacion
 * @param args vector de Object* vacio.
 */
Object* operator*(const std::vector<Object*>& args);

/** Metodo nativo para efectuar la suma
 * @param args vector de Object* vacio.
 * \retval Object con el resultado de la operacion. Se opera sobre
 * el primer operando, es decir, sobre el llamante del metodo.
 */
Object* operator+(const std::vector<Object*>& args);

/** Metodo nativo para efectuar la resta
 * @param args vector de Object* vacio.
 * \retval Object con el resultado de la operacion. Se opera sobre
 * el primer operando, es decir, sobre el llamante del metodo.
 */
Object* operator-(const std::vector<Object*>& args);

/** Metodo nativo para efectuar la division
 * @param args vector de Object* vacio.
 * \retval Object con el resultado de la operacion. Se opera sobre
 * el primer operando, es decir, sobre el llamante del metodo.
 */
Object* operator/(const std::vector<Object*>& args);

/** Metodo nativo para chequear igualdad
 * @param args vector de Object* vacio.
 * \retval Object con el resultado de la operacion. Se opera sobre
 * el primer operando, es decir, sobre el llamante del metodo.
```

```
     */
    Object* operator==(const std::vector<Object*>& args);

    /** Metodo nativo para chequear desigualdad
     * @param args vector de Object* vacio.
     * \retval Object con el resultado de la operacion. Se opera sobre
     * el primer operando, es decir, sobre el llamante del metodo.
     */
    Object* operator!=(const std::vector<Object*>& args);

public:
    /// Constructor que sirve para construir el objeto lobby.
    Object();

    /// Constructor de los objetos que no son Lobby.
    Object(Object* lobby);

    /// Destructor
    ~Object();

    /// Constructor copia
    Object(const Object& __object);

    /** Constructor por movimiento deshabilitado
     *
     */
    Object(Object&& obj) = delete;

    /** Operador de asignacion deshabilitado
     *
     */
    Object& operator=(const Object& _object) = delete;

    /** Operador de asignacion por moviemiento deshabilitado
     *
     */
    Object& operator=(Object&&) = delete;

    /** Devuelve todos los slots que tiene el objeto en forma de diccionario.
     *
     */
    slot_map getSlots() const;

    /** Devuelve todo los metodos nativos que tiene el objeto en forma de
     * diccionario.
     *
     */
    std::map<std::string, fpointTuple> getNativeMethods() const;

    /** Agrega el slot con los datos pasados como parametros.
     * @param name nombre del slot.
     * @param obj puntero a Object* para guardar en el slot.
     * @param _mutable booleano que indica si es mutable o no.
     * @param isParentSlot booleano que indica si es parent slot o no.
     * @param isArgument booleano que indica si es argumento o no.
     *
     */
```

```
Object* addSlot(std::string &name, Object* obj, bool _mutable,
        bool isParentSlot, bool isArgument);

/** Borra el slot especificado
 * @param name indica el nombre del slot a borrar.
 */
Object* removeSlot(std::string &name);

/** Agrega como code segment del objeto la cadena que se la pasa
 * @param code nuevo codigo para reemplazar en el code segment.
 */
void setCodeSegment(const std::string &code);

/** Devuelve el code segment del objeto.
 *
 */
std::string getCodeSegment() const;

/** Setea el nombre del objeto
 * @param name con el nombre del objeto para cambiar.
 */
void setName(const std::string &name);

/** Devuelve el nombre del objeto.
 *
 */
std::string getName() const;

/** Determina si el slot buscado del objeto es un DataObject o un
    MethodObject.
 * @param messageName nombre del slot.
 * \retval true si es data object
 * \retval false si no lo es.
 */
bool isDataObject(std::string &messageName);

/** Determina si es un DataObject o un MethodObject.
 * @param messageName nombre del slot.
 * \retval true si es data object
 * \retval false si no lo es.
 */
bool isDataObject();

/** Determina si es un metodo nativo.
 * @param messageName nombre del slot.
 * \retval true si es un metodo nativo
 * \retval false si no lo es.
 */
bool isNativeMethod(std::string &messageName);

/** Metodo principal que sirve para recibir mensajes. Devuelve un Object*
 * @param messageName nombre del slot que va a recibir el mensaje.
 * @param args vector con Object* con los argumentos para pasar.
 * @param clone indica si hay que clonar o no.
 * \retval Object con el slot solicitado con los argumentos procesados.
 */
Object* recvMessage(std::string &messageName, std::vector<Object*> args,
```

```
            bool clone);

    /** Metodo nativo para invocar el garbage collector.
     * @param args vector de Object* vacio.
     */
    Object* collect(const std::vector<Object*>& args);

    /** Habilita el metodo nativo.
     * @param methodName nombre del metodo a habilitar.
     * \retval Object con el resultado de la operacion. Se opera sobre
     * el primer operando, es decir, sobre el llamante del metodo.
     */
    void enableNativeMethod(std::string methodName);

    /** Deshabilita el metodo nativo.
     * @param methodName nombre del metodo a habilitar.
     *
     */
    void disableNativeMethod(std::string methodName);

    /** Agrega un objeto a la lista de objetos creados.
     * Solo debe invocarse desde lobby.
     * @param obj objeto a agregar.
     */
    void addCreatedObject(Object *obj);

    /** Devuelve un objeto. Solo debe invocarse desde Lobby.
     * @param id id del objeto a buscar.
     */
    Object* findObjectById(uint32_t id);

    /** Devuelve el id del objeto.
     *
     */
    uint32_t getId() const;

    /** Cambia el atributo de mutabilidad de un slot.
     * @param slotName nombre del slot a modificar
     */
    void swapSlotMutability(const std::string &slotName);

    /** Cambia el atributo de objeto primitivo.
     * @param newValue nuevo valor.
     */
    void setPrimitive(const bool newValue);

    /** Devuelve un booleano que indica si el objeto es primitivo o no.
     *
     */
    bool getPrimitive() const;
};

#endif /* COMMON_OBJECT_H_ */
```

36 server_object.cpp

```
#include "server_object.h"

#include <map>
#include <tuple>
#include <iostream>
#include "common_define.h"

Object::Object(Object* lobby) {

    this->lobby = lobby;
    configureNativeMethods();
}

Object::Object() {
    this->lobby = this;
    configureNativeMethods();
}

Object::Object(const Object& __object) {
    this->lobby = __object.lobby;

    // Recorro los slots de __object
    for (auto it = __object.slots.begin(); it != __object.slots.end(); ++it) {
        std::string name = it->first;
        slot_t tuple = it->second;
        Object* obj;

        bool isParentSlot = std::get<2>(tuple);
        if (isParentSlot) {
            obj = (Object*) std::get<0>(tuple);
        } else {
            Object tmpObj = *(Object*) std::get<0>(tuple);
            obj = new Object(tmpObj);

            auto tuple = std::make_tuple(obj, false);
            // Creo una tupla con el nuevo objeto creado
            // Despues lo inserto en la lista de objetos creados de lobby.
            lobby->createdObjects.insert(
                    std::make_pair(lobby->idCounter, tuple));
            id = lobby->idCounter;
            lobby->idCounter++;

        }
        std::get<0>(tuple) = obj;

        this->slots.insert(std::make_pair(name, tuple));
    }

    this->nativeMethods = __object.nativeMethods;
    this->codeSegment = __object.codeSegment;
    this->isPrimitive = __object.isPrimitive;
}

Object::~Object() {
    slots.clear();
```

```
    nativeMethods.clear();

    std::stack<Object*> objects;

    for (auto it = createdObjects.begin(); it != createdObjects.end(); ++it) {
        auto tuple = it->second;
        objects.push(std::get<0>(tuple));
    }
    createdObjects.clear();
    while (objects.size() > 0) {
        Object *obj = objects.top();
        delete obj;

        objects.pop();
    }
}

void Object::configureNativeMethods() {
    this->nativeMethods.insert(
            std::make_pair(PRINTOBJ_METHOD,
                    std::make_tuple(&Object::printObj, true)));
    this->nativeMethods.insert(
            std::make_pair(PRINT_METHOD,
                    std::make_tuple(&Object::print, false)));
    this->nativeMethods.insert(
            std::make_pair(OP_SUMA,
                    std::make_tuple(&Object::operator+, false)));
    this->nativeMethods.insert(
            std::make_pair(OP_RESTA,
                    std::make_tuple(&Object::operator-, false)));
    this->nativeMethods.insert(
            std::make_pair(OP_MULTIPLICACION,
                    std::make_tuple(&Object::operator*, false)));
    this->nativeMethods.insert(
            std::make_pair(OP_DIVISION,
                    std::make_tuple(&Object::operator/, false)));
    this->nativeMethods.insert(
            std::make_pair(OP_IGUAL,
                    std::make_tuple(&Object::operator==, false)));
    this->nativeMethods.insert(
            std::make_pair(OP_DISTINTO,
                    std::make_tuple(&Object::operator!=, false)));
    this->nativeMethods.insert(
            std::make_pair(ADD_SLOTS_METHOD,
                    std::make_tuple(&Object::_AddSlots, true)));
    this->nativeMethods.insert(
            std::make_pair(REMOVE_SLOTS_METHOD,
                    std::make_tuple(&Object::_RemoveSlots, true)));
    this->nativeMethods.insert(
            std::make_pair(CLONE_METHOD,
                    std::make_tuple(&Object::clone, true)));
    this->nativeMethods.insert(
            std::make_pair(COLLECT_METHOD,
                    std::make_tuple(&Object::collect, false)));
}

Object::slot_map Object::getSlots() const {
```

```
    return slots;
}

std::map<std::string, Object::fpointTuple> Object::getNativeMethods() const {
    return nativeMethods;
}

Object* Object::addSlot(std::string &name, Object* obj, bool _mutable,
        bool isParentSlot, bool isArgument) {
    this->slots.insert(
            std::make_pair(name,
                    std::make_tuple(obj, _mutable, isParentSlot, isArgument)));
    return this;
}

Object* Object::removeSlot(std::string &name) {
    auto _it = slots.find(name);
    if (_it != slots.end()) {
        slots.erase(_it);
    } else
        throw std::runtime_error("No existe el slot que se quiere borrar.");
    return this;
}

std::string Object::getCodeSegment() const {
    return this->codeSegment;
}

void Object::setCodeSegment(const std::string &code) {
    this->codeSegment = code;
}

std::string Object::getName() const {
    return this->name;
}

void Object::setName(const std::string &name) {
    this->name = name;
}

uint32_t Object::getId() const {
    return id;
}

void Object::setPrimitive(const bool newValue) {
    isPrimitive = newValue;
}

bool Object::getPrimitive() const {
    return isPrimitive;
}

bool Object::findObject(std::string name, Object* &returnValue,
        delegate& function) const {

    returnValue = nullptr;
    function = nullptr;
```

```
    // Primero me fijo en mis slots
    auto it = slots.find(name);
    if (it != slots.end()) {
        // Significa que lo encontre
        returnValue = (Object*) std::get<0>(it->second);
        return true;
    }

    slot_map parents = getParentSlots();
    for (auto parentSlot_it = parents.begin(); parentSlot_it != parents.end();
            ++parentSlot_it) {
        Object* pslot = (Object*) std::get<0>(parentSlot_it->second);
        if (pslot->findObject(name, returnValue, function))
            return true;
    }

    // No lo encontro en sus slots. Busco en los metodos nativos
    auto it_native = nativeMethods.find(name);
    if (it_native != nativeMethods.end()) {
        // Significa que lo encontre
        fpointTuple tuple = it_native->second;

        // Pregunto si esta habilitado
        if (std::get<1>(tuple)) {
            // llamo a la funcion apuntada
            function = std::get<0>(tuple);
            return true;
        }
    }

    return false;
}

bool Object::isDataObject(std::string &messageName) {
    // Primero verifico que el slot este en la lista de los slots,
    // esto es que este agregado o que se haya sobrecargado un metodo
    // nativo.
    Object* obj;
    delegate func;
    bool retval = findObject(messageName, obj, func);

    if (retval && obj) {
        if (obj->isPrimitive)
            return true;

        return (obj->codeSegment.size() == 0);
    } else if (retval && func) {
        return false;
    } else {
        std::string error = "El␣slot␣" + messageName + "␣no␣fue␣encontrado.";
        throw std::runtime_error(error);
    }
}

bool Object::isDataObject() {
    if (isPrimitive)
```

```
        return true;

    return (codeSegment.size() == 0);
}

bool Object::isNativeMethod(std::string &messageName) {
    Object* obj;
    delegate func;
    bool retval = findObject(messageName, obj, func);

    return (retval && func);
}

Object* Object::recvMessage(std::string &messageName, std::vector<Object*> args,
        bool clone) {
    Object* message;
    delegate fpointer;

    // Busco el slot solicitado
    if (!findObject(messageName, message, fpointer)) {
        std::string error = messageName + "␣no␣encontrado.";
        throw std::runtime_error(error);
    }

    // si fpointer no es nullptr, es un puntero a una funcion nativa
    if (fpointer != nullptr) {
        if (clone)
            message = this->clone(std::vector<Object*> { });
        else
            message = this;

        // Devuelvo el resultado de la llamada a la funcion
        // nativa.
        return (message->*fpointer)(args);
    }

    // Si hay que clonar, clono el mensaje para luego modificarle
    // los argumentos
    if (clone)
        message = message->clone(std::vector<Object*> { });

    // Una vez que tengo el objeto, necesito los argumentos, si es que tiene
    // y les cambio el valor con los argumentos que se pasaron como parametro
    slot_map object_slots = message->slots;

    // Contador de argumentos
    uint32_t argsCount = 0;
    for (auto object_slots_it = object_slots.begin();
            object_slots_it != object_slots.end(); ++object_slots_it) {
        // Verifico que la cant de argumentos sean iguales que los pasados,
        // para ahorrar ciclos.
        if (argsCount == args.size())
            break;

        // Verifico que sea argumento y que el slot sea mutable para poder
            modificarlo
        bool __isMutable = std::get<1>(object_slots_it->second);
```

```
        bool __isArg = std::get<3>(object_slots_it->second);

        if (__isArg && __isMutable) {
            slot_t tuple = object_slots_it->second;

            Object *__object = ((Object*) std::get<0>(tuple));
            __object = args[argsCount];
            std::get<0>(tuple) = __object;

            // actualizo el valor del mapa
            object_slots_it->second = tuple;
            argsCount++;
        }
    }

    // Si hay argumentos significa que hay que cambiar
    // los punteros de los argumentos.
    if (argsCount != 0) {
        message->slots = object_slots;
        return message;
    }

    // Busco en los slots el mensaje, lo que se esta implementando aca
    // es un cambio de valor de un objeto mutable.
    // Si no es mutable, genera excepcion.
    auto it = slots.find(messageName);
    if (it != slots.end() && args.size() > 0) {
        slot_t tuple = it->second;

        if (std::get<1>(tuple)) {
            std::get<0>(tuple) = args[0];
            it->second = tuple;

            return args[0];
        } else {
            std::string error = "El slot no es mutable";
            throw std::runtime_error(error);
        }
    }

    return message;
}

Object::slot_map Object::getParentSlots() const {
    slot_map parentSlots;
    for (auto it = slots.begin(); it != slots.end(); ++it) {
        if (std::get<2>(it->second)) {
            if (std::get<0>(it->second) != this)
                parentSlots.insert(std::make_pair(it->first, it->second));
        }
    }
    return parentSlots;
}

void Object::enableNativeMethod(std::string methodName) {
    auto fpoint = nativeMethods.find(methodName);
    if (fpoint == nativeMethods.end()) {
```

```
        std::string error = "No existe el mensaje ";
        error += methodName;
        throw std::runtime_error(error);
    }
    fpointTuple tuple = fpoint->second;
    std::get<1>(tuple) = true;
    fpoint->second = tuple;
}

void Object::disableNativeMethod(std::string methodName) {
    auto fpoint = nativeMethods.find(methodName);
    if (fpoint == nativeMethods.end()) {
        std::string error = "No existe el mensaje ";
        error += methodName;
        throw std::runtime_error(error);
    }
    fpointTuple tuple = fpoint->second;
    std::get<1>(tuple) = false;
    fpoint->second = tuple;
}

void Object::addCreatedObject(Object *obj) {
    auto tuple = std::make_tuple(obj, false);
    lobby->createdObjects.insert(std::make_pair(lobby->idCounter, tuple));
    obj->id = lobby->idCounter;
    lobby->idCounter++;
}

Object* Object::findObjectById(uint32_t id) {
    if (id == ID_LOBBY)
        return lobby;
    auto it = createdObjects.find(id);

    if (it == createdObjects.end()) {
        std::string error = "No existe un objeto con el id "
                + std::to_string(id);
        throw std::runtime_error(error);
    } else
        return std::get<0>(it->second);
}

void Object::collect_internal() {
    // Recorro todos los slots,
    for (auto object_slots_it = slots.begin(); object_slots_it != slots.end();
            ++object_slots_it) {
        slot_t tuple = object_slots_it->second;

        // Si no es parent slot, llamo recursivamente
        // a collect_internal para ir marcando los objetos
        // que no tienen acceso desde lobby.
        if (!std::get<2>(tuple)) {
            Object* slotObj = std::get<0>(tuple);
            slotObj->collect_internal();

            // busco el objeto (por su direccion de memoria) y
            // lo marco, indicando que se puede acceder desde lobby.
            for (auto it = lobby->createdObjects.begin();
```

```
                    it != lobby->createdObjects.end(); ++it) {
                auto _tuple = it->second;
                Object* pointer = std::get<0>(_tuple);
                if (pointer == slotObj) {
                    std::get<1>(_tuple) = true;
                    it->second = _tuple;
                }
            }
        }
    }
}

void Object::swapSlotMutability(const std::string& slotName) {

    auto it = slots.find(slotName);
    if (it != slots.end()) {
        auto tuple = it->second;

        bool _mutable = std::get<1>(tuple);

        std::get<1>(tuple) = !_mutable;
        it->second = tuple;
    } else {
        throw std::runtime_error("El slot buscado no existe");
    }
}

// Funciones nativas
Object* Object::_AddSlots(const std::vector<Object*>& args) {
    //Recorro obj y agrego sus slots
    Object *obj = args[0];

    for (auto it = obj->slots.begin(); it != obj->slots.end(); ++it) {
        this->slots.insert(std::make_pair(it->first, it->second));
    }
    return this;
}

Object* Object::_RemoveSlots(const std::vector<Object*>& args) {
    Object *obj = args[0];

    for (auto it = obj->slots.begin(); it != obj->slots.end(); ++it) {
        std::string name = it->first;
        auto _it = slots.find(name);
        if (_it != slots.end()) {
            slots.erase(_it);
        } else {
            disableNativeMethod(name);
        }
    }
    return this;
}

Object* Object::clone(const std::vector<Object *> &args) {
    Object* obj = new Object(*this);

    auto tuple = std::make_tuple(obj, false);
```

```
    lobby->createdObjects.insert(std::make_pair(lobby->idCounter, tuple));
    obj->id = lobby->idCounter;
    lobby->idCounter++;
    return obj;
}

Object* Object::collect(const std::vector<Object*>& args) {
    collect_internal();

    std::vector<uint32_t> deleteObjects;
    // Recorro los objetos creados, el unico objeto que
    // va a tener es lobby. Reviso que no esten marcados
    // los guardo en un vector para luego hacerles el delete.
    for (auto it = createdObjects.begin(); it != createdObjects.end(); ++it) {
        auto tuple = it->second;
        if (!std::get<1>(tuple)) {
            deleteObjects.push_back(it->first);
        }
    }

    for (auto id : deleteObjects) {
        Object* obj = std::get<0>((createdObjects.find(id))->second);
        createdObjects.erase(id);
        delete obj;
    }

    deleteObjects.clear();

    // Reconfiguro la lista para que vuelva a funcionar el GC
    // en una proxima iteracion.
    for (auto it = createdObjects.begin(); it != createdObjects.end(); ++it) {
        auto _tuple = it->second;
        std::get<1>(_tuple) = true;

    }
    return this;
}

Object* Object::printObj(const std::vector<Object*>& args) {
    std::cout << this << "␣" << name << ":␣";
    std::cout << P_LEFT << SLOT_LIST_SEP;

//Escribe los slots (metodos no nativos)
    for (auto _it = slots.begin(); _it != slots.end(); ++_it) {
        std::string slotName = _it->first;
        slot_t slot = _it->second;

        bool esMutable = std::get<1>(slot);
        bool esParent = std::get<2>(slot);
        bool esArgument = std::get<3>(slot);

        std::cout << "␣";

        if (esArgument)
            std::cout << OP_ARG;

        std::cout << slotName;
```

```
        if (esParent)
            std::cout << OP_PARENT;

        if (esMutable)
            std::cout << "␣" << OP_SLOT_MUTABLE << "␣";
        else
            std::cout << "␣" << OP_SLOT_INMUTABLE << "␣";
        Object* dirObj = (Object*) std::get<0>(slot);
        std::cout << dirObj;
        std::cout << PUNTO;
    }

//Escribe los slots nativos (metodos nativos)
    for (auto _it = nativeMethods.begin(); _it != nativeMethods.end(); ++_it) {
        std::string slotNameNative = _it->first;
        fpointTuple tuple = _it->second;
        if (std::get<1>(tuple))
            std::cout << "␣<" << slotNameNative << ">" << PUNTO;
    }

    std::cout << "␣" << SLOT_LIST_SEP << "␣";
    std::cout << codeSegment << "␣" << P_RIGHT << std::endl;

    for (auto _it = slots.begin(); _it != slots.end(); ++_it) {
        std::string slotName = _it->first;
        slot_t slot = _it->second;
        Object* dirObj;
        dirObj = (Object*) std::get<0>(slot);
        // Es distinto de this para el caso de lobby
        bool esParent = std::get<2>(slot);
        if (dirObj != nullptr && dirObj != this && !esParent)
            dirObj->printObj(std::vector<Object*> { });
        else if (dirObj == nullptr)
            std::cout << "ERROR:␣El␣Slot␣no␣apunta␣a␣ningun␣objeto."
                    << std::endl;
    }
    return this;
}

Object* Object::print(const std::vector<Object*>& args) {
    std::string _codeSegment = codeSegment.substr(0, codeSegment.size() - 1);

    for (uint32_t i = 0; i < _codeSegment.size(); i++) {
        size_t pos = _codeSegment.find('\'', i);
        if (pos != std::string::npos)
            _codeSegment.replace(pos, 1, "");
        else
            break;
    }

    if (_codeSegment == "\\n") {
        std::cout << std::endl;
    } else {
        std::cout << _codeSegment;
    }
```

```
    return this;
}

Object* Object::operator*(const std::vector<Object*>& args) {
    Object *first = (Object*) args[0];
    std::string strCodeSegment = this->codeSegment.substr(0,
            codeSegment.size() - 1);
    std::string argCodeSegment = first->codeSegment.substr(0,
            first->codeSegment.size() - 1);

    try {
        float number = std::stof(strCodeSegment);
        float operand = std::stof(argCodeSegment);
        codeSegment = std::to_string((int) (number * operand)) + PUNTO;
    } catch(...) {
        std::string err = "Operacion␣no␣valida␣porque␣los␣objetos␣no␣";
        err += "son␣compatibles.";
        throw std::runtime_error(err);
    }
    return this;
}

Object* Object::operator+(const std::vector<Object*>& args) {
    Object *first = (Object*) args[0];
    std::string strCodeSegment = this->codeSegment.substr(0,
            codeSegment.size() - 1);
    std::string argCodeSegment = first->codeSegment.substr(0,
            first->codeSegment.size() - 1);

    try {
        float number = std::stof(strCodeSegment);
        float operand = std::stof(argCodeSegment);
        codeSegment = std::to_string((int) (number + operand)) + PUNTO;
    } catch(...) {
        std::string err = "Operacion␣no␣valida␣porque␣los␣objetos␣no␣";
        err += "son␣compatibles.";
        throw std::runtime_error(err);
    }
    return this;
}

Object* Object::operator-(const std::vector<Object*>& args) {
    Object *first = (Object*) args[0];
    std::string strCodeSegment = this->codeSegment.substr(0,
            codeSegment.size() - 1);
    std::string argCodeSegment = first->codeSegment.substr(0,
            first->codeSegment.size() - 1);

    try {
        float number = std::stof(strCodeSegment);
        float operand = std::stof(argCodeSegment);
        codeSegment = std::to_string((int) (number - operand)) + PUNTO;
    } catch(...) {
        std::string err = "Operacion␣no␣valida␣porque␣los␣objetos␣no␣";
        err += "son␣compatibles.";
        throw std::runtime_error(err);
    }
```

```
    return this;
}

Object* Object::operator/(const std::vector<Object*>& args) {
    Object *first = (Object*) args[0];
    std::string strCodeSegment = this->codeSegment.substr(0,
            codeSegment.size() - 1);
    std::string argCodeSegment = first->codeSegment.substr(0,
            first->codeSegment.size() - 1);
    try {
        float number = std::stof(strCodeSegment);
        float operand = std::stof(argCodeSegment);
        codeSegment = std::to_string((int) (number / operand)) + PUNTO;
    } catch(...) {
        std::string err = "Operacion␣no␣valida␣porque␣los␣objetos␣no␣";
        err += "son␣compatibles.";
        throw std::runtime_error(err);
    }
    return this;
}

Object* Object::operator==(const std::vector<Object*>& args) {
    Object *first = (Object*) args[0];
    std::string strCodeSegment = this->codeSegment.substr(0,
            codeSegment.size() - 1);
    std::string argCodeSegment = first->codeSegment.substr(0,
            first->codeSegment.size() - 1);

    bool retVal = true;

    if (strCodeSegment.front() == '\'' && strCodeSegment.back() == '\''
            && argCodeSegment.front() == '\''
            && argCodeSegment.back() == '\'') {
        retVal = (strCodeSegment == argCodeSegment);
    } else {
        float number = std::stof(strCodeSegment);
        float operand = std::stof(argCodeSegment);
        retVal = (number == operand);
    }

    slots.clear();

    std::string _op;
    _op = OP_SUMA;
    disableNativeMethod(OP_SUMA);
    disableNativeMethod(OP_RESTA);
    disableNativeMethod(OP_MULTIPLICACION);
    disableNativeMethod(OP_DIVISION);
    codeSegment = (retVal ? TRUE_STR : FALSE_STR) + PUNTO;
    return this;
}

Object* Object::operator!=(const std::vector<Object*>& args) {
    Object* obj = this->operator==(args);
    std::string strCodeSegment = obj->codeSegment.substr(0,
            codeSegment.size() - 1);
```

```
    if (strCodeSegment == TRUE_STR)
        obj->codeSegment = FALSE_STR;
    else
        obj->codeSegment = TRUE_STR;

    obj->codeSegment += PUNTO;

    return obj;
}
```

37 server_virtual_machine.h

```
#ifndef SERVER_VIRTUALMACHINE_H_
#define SERVER_VIRTUALMACHINE_H_

#include <stack>
#include "server_object.h"

/** Es la encargada de crear y almacenarlos en una pila en el objeto lobby.
 * Tanto la maquina virtual como el objeto lobby son unicos por workspace.
 *
 */
class VirtualMachine {
private:
    Object* lobby = nullptr;

public:
    VirtualMachine(const VirtualMachine&) = delete;
    VirtualMachine(VirtualMachine&&) = delete;
    VirtualMachine& operator=(const VirtualMachine&) = delete;
    VirtualMachine& operator=(VirtualMachine&&) = delete;

    /** Constructor
     *
     */
    VirtualMachine();
    /** Crea un objeto primitivo nil
     *
     */
    Object *createNil();
    /** Crea un objeto primitivo string
     * @param strString cadena del objeto
     */
    Object *createString(std::string &strString);
    /** Crea un objeto primitivo number
     * @param number n mero del objeto
     */
    Object *createNumber(float number);
    /** Crea un objeto primitivo booleano
     * @param value estado del objeto booleano
     */
    Object *createBoolean(bool value);
    /** Crea un objeto no primitivo vacio
     */
    Object* createEmptyObject();
    /** Busca un objeto por su ID y lo retorna
     * @param id id del objeto buscado
     */
    Object* findObjectById(uint32_t id);

    /** Setea el lobby de la maquina virtual creado por el workspace
     * @param lobby objeto lobby creado por el workspace
     */
    void setLobby(Object* lobby);
};

#endif /* SERVER_VIRTUALMACHINE_H_ */
```

38 server_virtual_machine.cpp

```
#include "server_virtual_machine.h"

#include <string>
#include <iostream>
#include "common_define.h"

VirtualMachine::VirtualMachine() {
}

void VirtualMachine::setLobby(Object* lobby) {
    this->lobby = lobby;
}

Object* VirtualMachine::createString(std::string &strString) {
    Object *obj = new Object(lobby);
    obj->setPrimitive(true);
    obj->setName(STRING_OBJ);
    obj->setCodeSegment(strString + PUNTO);
    obj->enableNativeMethod(PRINT_METHOD);
    obj->enableNativeMethod(OP_IGUAL);
    obj->enableNativeMethod(OP_DISTINTO);
    lobby->addCreatedObject(obj);
    return obj;
}

Object* VirtualMachine::createNumber(float number) {
    Object *obj = new Object(lobby);
    obj->setPrimitive(true);
    obj->setName(NUMBER_OBJ);
    obj->setCodeSegment(std::to_string((int) number) + PUNTO);
    obj->enableNativeMethod(PRINT_METHOD);
    obj->enableNativeMethod(OP_SUMA);
    obj->enableNativeMethod(OP_RESTA);
    obj->enableNativeMethod(OP_MULTIPLICACION);
    obj->enableNativeMethod(OP_DIVISION);
    obj->enableNativeMethod(OP_IGUAL);
    obj->enableNativeMethod(OP_DISTINTO);
    lobby->addCreatedObject(obj);
    return obj;
}

Object* VirtualMachine::createNil() {
    Object *obj = new Object(lobby);
    obj->setPrimitive(true);
    obj->setName(NIL_OBJ);
    obj->setCodeSegment(NIL + PUNTO);
    obj->enableNativeMethod(PRINT_METHOD);
    obj->enableNativeMethod(OP_IGUAL);
    obj->enableNativeMethod(OP_DISTINTO);
    lobby->addCreatedObject(obj);
    return obj;
}

Object* VirtualMachine::createEmptyObject() {
    Object *obj = new Object(lobby);
```

```
    obj->setName(COMPLEX_OBJ);
    lobby->addCreatedObject(obj);
    return obj;
}

Object* VirtualMachine::createBoolean(bool value) {
    Object *obj = new Object(lobby);
    obj->setPrimitive(true);
    obj->setName(BOOLEAN_OBJ);
    obj->enableNativeMethod(PRINT_METHOD);
    if (value)
        obj->setCodeSegment(TRUE_STR + PUNTO);
    else
        obj->setCodeSegment(FALSE_STR + PUNTO);
    lobby->addCreatedObject(obj);
    return obj;
}

Object* VirtualMachine::findObjectById(uint32_t id) {
    return lobby->findObjectById(id);
}
```

39 server_mode_selector.h

```
#ifndef MODESELECTOR_H_
#define MODESELECTOR_H_

#include "server_accepter.h"
#include "common_define.h"
#include <fstream>

/** Es la clase selectora para los dos modos del servidor.
 * El modo servidor propiamente dicho y el modo para levantar
 * archivos locales con codigo self.
 *
 */
class ModeSelector {
public:
    /** Constructor del modo server
     * @param port puerto de escucha del servidor para aceptar conexiones.
     */
    ModeSelector(int port);

    /** Constructor del modo archivo
     * @param filename nombre del archivo con el script en c digo self
     * que se desea ejecutar.
     */
    ModeSelector(std::string filename);

    /** Constructor por copia deshabilitado
     */
    ModeSelector(const ModeSelector&) = delete;

    /** Constructor por movimiento deshabilitado
     */
    ModeSelector(ModeSelector&&) = delete;

    /** Operador de asignacion deshabilitado
     */
    ModeSelector& operator=(const ModeSelector&) = delete;

    /** Operador de asignacion por moviemiento deshabilitado
     */
    ModeSelector& operator=(ModeSelector&&) = delete;

private:
    /** Metodo que interrumpe al aceptador y joinea el hilo que se abrio
     * @param accepter Objeto aceptador de conexiones.
     */
    void exitRoutine(Accepter* accepter);
};

#endif /* MODESELECTOR_H_ */
```

40 server_mode_selector.cpp

```
#include "server_mode_selector.h"
#include "server_workspace.h"

ModeSelector::ModeSelector(int port) {
    try {
        Server server;
        Accepter *accepter = new Accepter(port, server);
        // Este hilo va a estar corriendo durante toda la ejecucion del programa
        accepter->start();

        char c = '\0';
        while (c != SERVER_QUIT_CHAR) {
            std::cin.get(c);
        }
        exitRoutine(accepter);
        delete accepter;
    } catch (const std::runtime_error &e) {
        std::cout << "Error. " << std::endl << e.what() << std::endl;
    } catch (...) {
        std::cout << "Error desconocido." << std::endl << std::endl;
    }
}

ModeSelector::ModeSelector(std::string filename) {
    Workspace workspace;

    std::ifstream filein(filename);

    if (!filein.is_open())
        throw std::runtime_error("No se pudo abrir el archivo.");

    std::string script, x;
    while (filein >> x)
        script += x + " ";

    std::cout << script << std::endl;
    workspace.receive(workspace.getLobby(), script);
}

void ModeSelector::exitRoutine(Accepter* accepter) {
    // mando la se al para interrumpir el aceptador de conexiones
    accepter->interrupt();
    accepter->join();
}
```

41 server_parser_protocolo_servidor.h

```
#ifndef PARSER_PROTOCOLO_SERVIDOR_H_
#define PARSER_PROTOCOLO_SERVIDOR_H_

#include <string>
#include "server_object.h"

/** Esta clase se encarga de generar una cadena con el formato especificado
 * por protocolo en funcion de un objeto dado por el servidor para que la
 * misma sea enviada por el ProxyClient al cliente.
 *
 */
class ParserProtocoloServidor {
private:
    ///Objeto con el que se genera la cadena
    Object* obj;

public:
    /** Constructor
     * @param obj Objeto con el que se genera la cadena
     *
     */
    ParserProtocoloServidor(Object* obj);

    /** Constructor por copia deshabilitado
     */
    ParserProtocoloServidor(const ParserProtocoloServidor&) = delete;
    /** Constructor por movimiento deshabilitado
     */
    ParserProtocoloServidor(ParserProtocoloServidor&&) = delete;
    /** Operador de asignacion deshabilitado
     */
    ParserProtocoloServidor& operator=(const ParserProtocoloServidor&) = delete;
    /** Operador de asignacion por moviemiento deshabilitado
     */
    ParserProtocoloServidor& operator=(ParserProtocoloServidor&&) = delete;
    /** Genera la cadena por protocolo en funcion del objeto obj
     * y la retorna.
     */
    std::string getString();
};

#endif /* PARSER_PROTOCOLO_SERVIDOR_H_ */
```

42 server_parser_protocolo_servidor.cpp

```cpp
#include "server_parser_protocolo_servidor.h"
#include "common_define.h"

ParserProtocoloServidor::ParserProtocoloServidor(Object* obj) {
    this->obj = obj;
}

std::string ParserProtocoloServidor::getString() {
    std::string cad;
    cad += obj->getName();
    cad += CHAR_SEPARADOR;
    cad += obj->getCodeSegment();

    Object::slot_map slots = obj->getSlots();
    std::map<std::string, Object::fpointTuple> nativeMethods =
            obj->getNativeMethods();
    //Leemos los slots que apuntan a metodos nativos

    for (auto _it = nativeMethods.begin(); _it != nativeMethods.end(); ++_it) {
        std::string slotNameNative = _it->first;
        Object::fpointTuple tuple = _it->second;
        //Verificamos que el metodo nativo este activo para el objeto
        if (std::get<1>(tuple)) {
            cad += CHAR_SEPARADOR;
            cad += slotNameNative;

            //Indicamos que es metodo nativo
            cad += CHAR_SEPARADOR;
            cad += TRUE_BIN;

            //No es mutable
            cad += CHAR_SEPARADOR;
            cad += FALSE_BIN;

            //No es parent slot
            cad += CHAR_SEPARADOR;
            cad += FALSE_BIN;

            //No es argument slot
            cad += CHAR_SEPARADOR;
            cad += FALSE_BIN;

            //Nombre native method
            cad += CHAR_SEPARADOR;
            cad += NATIVE_METHOD;

            //Vista previa
            cad += CHAR_SEPARADOR;
            cad += COMPLEX_PREVIEW;
        }
    }

    //Leemos los slots que apuntan a objetos (metodos no nativos)
    for (auto _it = slots.begin(); _it != slots.end(); ++_it) {
        std::string slotName = _it->first;
```

```
        Object::slot_t slot = _it->second;
        bool esMutable = std::get<1>(slot);
        bool esParent = std::get<2>(slot);
        bool esArgument = std::get<3>(slot);

        cad += CHAR_SEPARADOR;
        cad += slotName;

        //Indicamos que NO es metodo nativo
        cad += CHAR_SEPARADOR;
        cad += FALSE_BIN;

        cad += CHAR_SEPARADOR;
        if (esMutable == false)
            cad += FALSE_BIN;
        else
            cad += TRUE_BIN;

        cad += CHAR_SEPARADOR;
        if (esParent == false)
            cad += FALSE_BIN;
        else
            cad += TRUE_BIN;

        cad += CHAR_SEPARADOR;
        if (esArgument == false)
            cad += FALSE_BIN;
        else
            cad += TRUE_BIN;

        Object* objSlot = (Object*) std::get<0>(slot);
        if (objSlot != nullptr) {
            cad += CHAR_SEPARADOR;
            cad += objSlot->getName();
            cad += CHAR_SEPARADOR;
            cad += objSlot->getCodeSegment();
        } else {
            cad += CHAR_SEPARADOR;
            cad += "";
            cad += CHAR_SEPARADOR;
            cad += "";
        }

        objSlot->getName();
    }
    return cad;
}
```