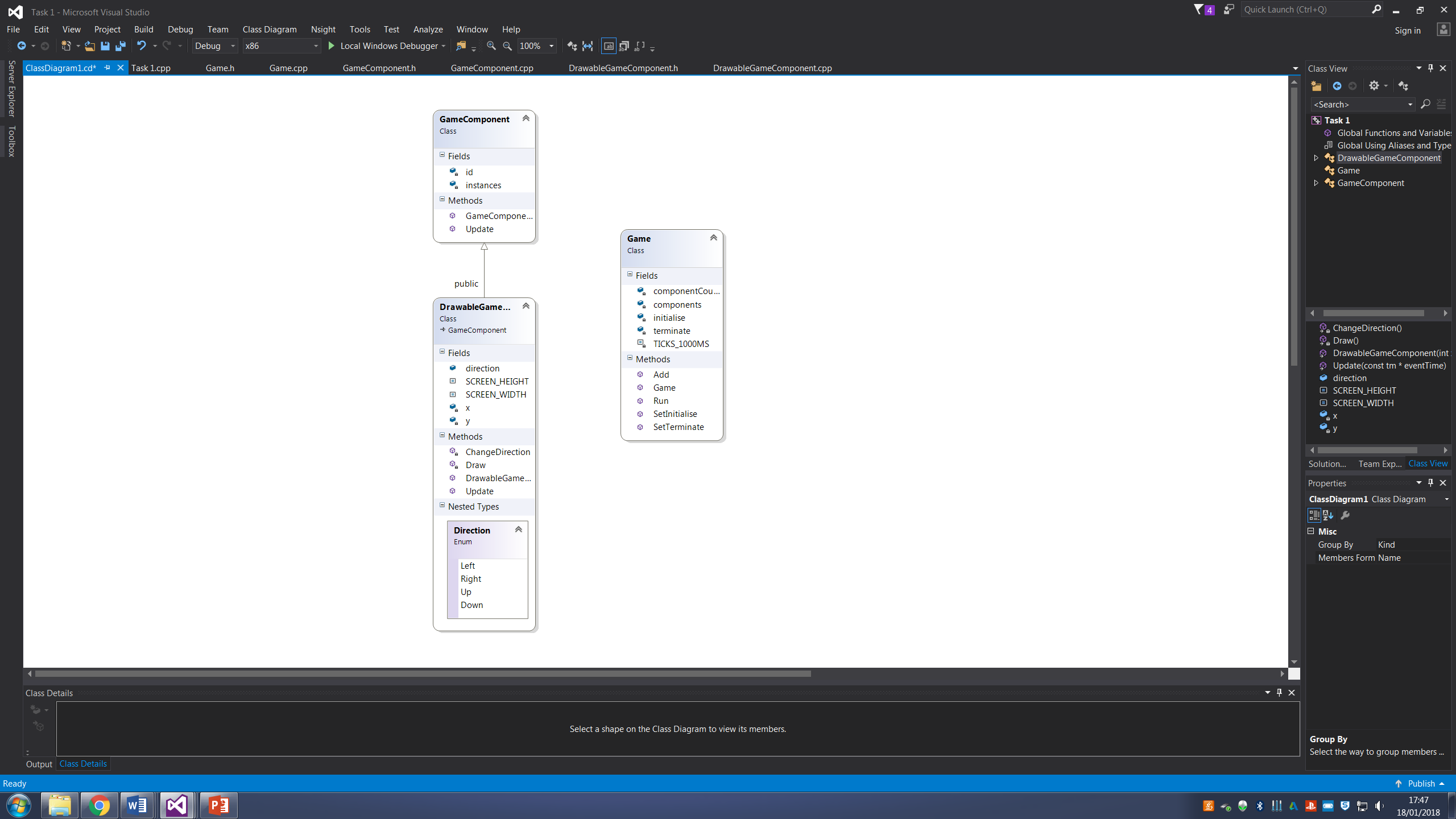
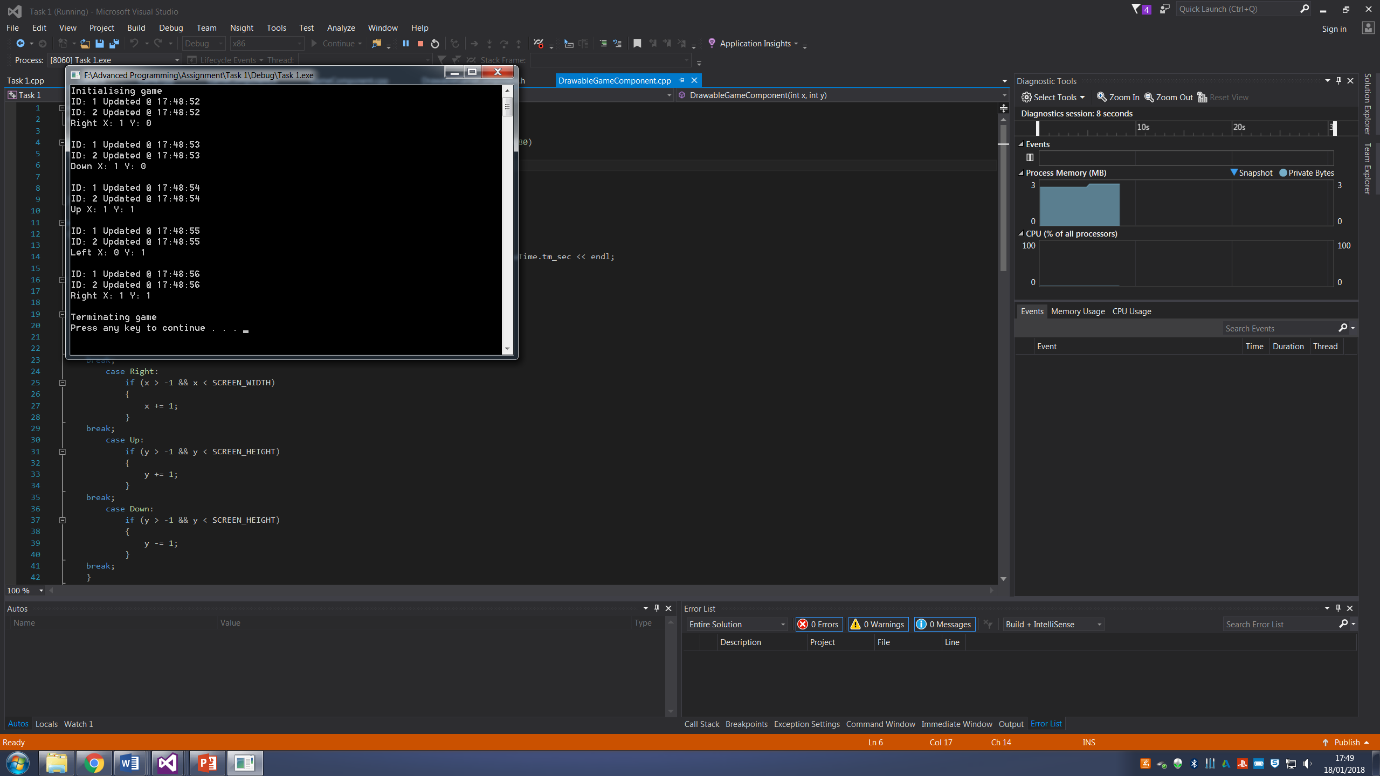
**Advanced Programming – CW1**

# **Task 1**





## **Task1.cpp**

// Task 1.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include "DrawableGameComponent.h"

#include "Game.h"

#include "GameComponent.h"

void Initialise()

{

cout << "Initialising game" << endl;

}

void Terminate()

{

cout << "Terminating game" << endl;

}

int main()

{

Game \*G = new Game(2);

G->SetInitialise(\*Initialise);

G->SetTerminate(\*Terminate);

G->Add(new GameComponent);

G->Add(new DrawableGameComponent(0, 0));

G->Run();

system("PAUSE");

return 0;

}

## **Game.h**

#pragma once

#include "GameComponent.h"

typedef void (\*FP)();

class Game

{

public:

void Add(GameComponent\*);

Game(int maxComponents);

void Run();

void SetInitialise(FP init);

void SetTerminate(FP term);

private:

int componentCount;

GameComponent\*\* components;

FP initialise;

FP terminate;

const int TICKS\_1000MS;

};

## **Game.cpp**

#include "stdafx.h"

#include "Game.h"

void Game::Add(GameComponent\* component)

{

components[componentCount] = component;

componentCount++;

}

Game::Game(int maxComponents) : TICKS\_1000MS(1000)

{

components = new GameComponent\*[maxComponents];

}

void Game::Run()

{

initialise();

\_\_time32\_t rawTime;

struct tm time;

for (int a = 0; a < 5; a++)

{

for (int i = 0; i < componentCount; i++)

{

\_time32(&rawTime);

\_localtime32\_s(&time, &rawTime);

components[i]->Update(&time);

}

Sleep(TICKS\_1000MS);

}

terminate();

}

void Game::SetInitialise(FP init)

{

initialise = init;

}

void Game::SetTerminate(FP term)

{

terminate = term;

}

## **GameComponent.h**

#pragma once

#include <iostream>

#include <iomanip>

#include <time.h>

#include <string>

#include <windows.h>

using namespace std;

class GameComponent

{

public:

GameComponent();

virtual void Update(const tm\*);

friend class DrawableGameComponent;

private:

int id = 0;

static int instances;

};

## **GameComponent.cpp**

#include "stdafx.h"

#include "GameComponent.h"

int GameComponent::instances = 0;

GameComponent::GameComponent()

{

instances++;

id = instances;

}

void GameComponent::Update(const tm\* eventTime)

{

struct tm uTime = \*eventTime;

cout << "ID: " << id << " Updated @ " << uTime.tm\_hour << ":" << uTime.tm\_min << ":" << uTime.tm\_sec << endl;

}

## **DrawableGameComponent.h**

#pragma once

#include "GameComponent.h"

class DrawableGameComponent : public GameComponent

{

public:

enum Direction {Left, Right, Up, Down};

Direction direction;

DrawableGameComponent(int x, int y);

const int SCREEN\_HEIGHT;

const int SCREEN\_WIDTH;

void Update(const tm\* eventTime);

private:

void ChangeDirection();

void Draw();

int x;

int y;

};

## **DrawableGameComponent.cpp**

#include "stdafx.h"

#include "DrawableGameComponent.h"

DrawableGameComponent::DrawableGameComponent(int x, int y) : SCREEN\_HEIGHT(20), SCREEN\_WIDTH(80)

{

this->x = 0;

this->y = 0;

this->direction = Right;

}

void DrawableGameComponent::Update(const tm \* eventTime)

{

struct tm uTime = \*eventTime;

cout << "ID: " << id << " Updated @ " << uTime.tm\_hour << ":" << uTime.tm\_min << ":" << uTime.tm\_sec << endl;

switch (direction)

{

case Left:

if (x > -1 && x < SCREEN\_WIDTH)

{

x -= 1;

}

break;

case Right:

if (x > -1 && x < SCREEN\_WIDTH)

{

x += 1;

}

break;

case Up:

if (y > -1 && y < SCREEN\_HEIGHT)

{

y += 1;

}

break;

case Down:

if (y > -1 && y < SCREEN\_HEIGHT)

{

y -= 1;

}

break;

}

if (x < 0)

{

x = 0;

}

if (y < 0)

{

y = 0;

}

Draw();

ChangeDirection();

}

void DrawableGameComponent::ChangeDirection()

{

Direction previousDirection = direction;

do

{

direction = static\_cast<Direction>(rand() % 4);

} while (previousDirection == direction);

}

void DrawableGameComponent::Draw()

{

switch (direction)

{

case Left:cout << "Left";

break;

case Right:cout << "Right";

break;

case Up:cout << "Up";

break;

case Down:cout << "Down";

break;

default: break;

}

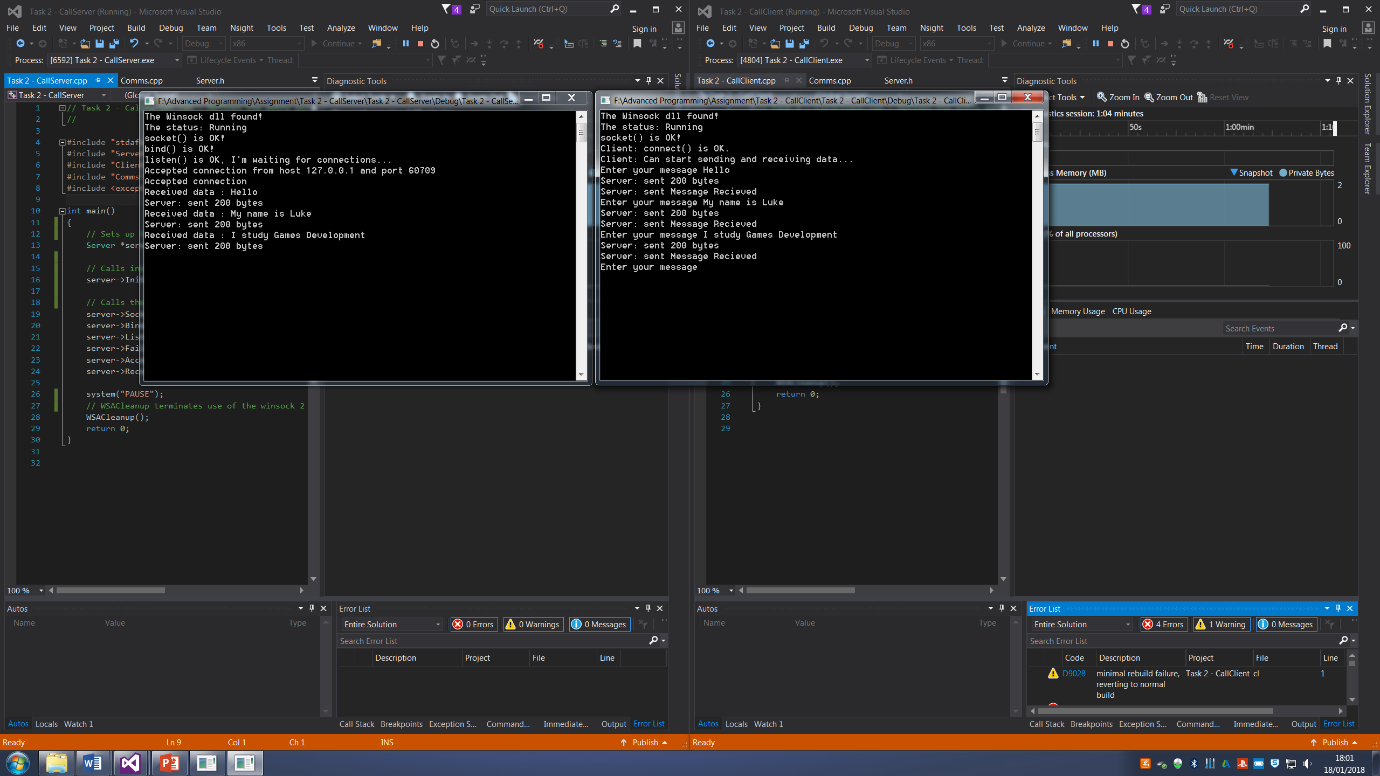
cout << " X: " << x << " Y: " << y << endl << endl;

}

# **Task 2**

CallClient Console

CallServer Console



## **Task 2 – CallServer.cpp**

// Task 2 - CallServer.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include "Server.h"

#include "Client.h"

#include "Comms.h"

#include <exception>

int main()

{

// Sets up the server script so main can call its functions

Server \*server = new Server();

// Calls initialise in comms through the server script

server->Initialise();

// Calls the rest of servers functions

server->Socket();

server->Bind();

server->Listen();

server->FailedConnection();

server->AcceptConnection();

server->Receive();

system("PAUSE");

// WSACleanup terminates use of the winsock 2

WSACleanup();

return 0;

}

## **Task 2 – CallClient.cpp**

// Task 2 - CallClient.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include "Server.h"

#include "Client.h"

#include "Comms.h"

#include <exception>

int main()

{

// Sets up the client script so main can call its functions

Client \*client = new Client();

// Calls initialise in comms through the client script

client->Initialise();

// Calls the rest of clients functions

client->Socket();

client->Connect();

client->Send();

system("PAUSE");

// WSACleanup terminates use of the winsock 2

WSACleanup();

return 0;

}

## **Comms.h**

#pragma once

#include "stdafx.h"

#include "winsock2.h"

#include "ws2tcpip.h"

#include <iostream>

#include <exception>

using namespace std;

class Comms

{

public:

Comms();

void Initialise();

protected:

// Variable declarations that are both used in client and server cpps

SOCKADDR\_STORAGE from;

int retval, fromlen, socket\_type;

char servstr[NI\_MAXSERV], hoststr[NI\_MAXHOST];

SOCKET serverSocket, clientSocket, acceptSocket;

int port = 55555;

WSADATA wsaData;

int wsaerr;

WORD wVersionRequested = MAKEWORD(2, 2);

sockaddr\_in service;

sockaddr\_in clientService;

char buffer[200];

};

// Exception class used in server and client cpp to throw a server send error

class ServerSendError : exception

{

public:

virtual const char\* what() const throw()

{

return "Server send error %ld.\n";

}

};

// Exception class used in server and client cpp to throw a socket error

class SocketError : exception

{

public:

virtual const char\* what() const throw()

{

return "Error at socket(): ";

}

};

## **Comms.cpp**

#include "stdafx.h"

#include "Comms.h"

Comms::Comms()

{

}

void Comms::Initialise()

{

// Initialises WSA

wsaerr = WSAStartup(wVersionRequested, &wsaData);

try

{

if (wsaerr != 0)

{

// throws error when wsaerr is not equal to 0

throw "The Winsock dll not found!";

}

else

{

cout << "The Winsock dll found!" << endl;

cout << "The status: " << wsaData.szSystemStatus << endl;

}

}

catch (char \* error)

{

cout << "Exception : " << error << endl;

}

catch (...)

{

cout << "Exception : " << endl;

}

}

## **Server.h**

#pragma once

#include "Comms.h"

// Uses Comms as a base class

class Server : public Comms

{

public:

// Initializes server functions

Server();

void Socket();

void Bind();

void Listen();

void FailedConnection();

void AcceptConnection();

void Receive();

};

## **Server.cpp**

#include "stdafx.h"

#include "Server.h"

Server::Server()

{

}

// This function creates a socket that is bound to a specific transport service provider

void Server::Socket()

{

serverSocket = INVALID\_SOCKET;

serverSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

try

{

if (serverSocket == INVALID\_SOCKET)

{

// Throws error to SocketError Class

throw SocketError(), WSAGetLastError();

WSACleanup();

}

else

{

cout << "socket() is OK!" << endl;

}

}

catch (SocketError& s) // Catches class error

{

cout << "Error: " << s.what() << endl;

}

catch (...) // Default catch

{

cout << "Error: " << endl;

}

}

// This function associates a local address with a socket

void Server::Bind()

{

service.sin\_family = AF\_INET;

InetPton(AF\_INET, \_T("127.0.0.1"), &service.sin\_addr.s\_addr);

service.sin\_port = htons(port);

try

{

if (bind(serverSocket, (SOCKADDR\*)&service, sizeof(service)) == SOCKET\_ERROR)

{

// Throws char followed by WSAGetLastError

throw "bind() failed: ", WSAGetLastError();

closesocket(serverSocket);

WSACleanup();

}

else

{

cout << "bind() is OK!" << endl;

}

}

catch (char \* error) // Catches char error

{

cout << "Exception : " << error << endl;

}

catch (...) // Default catch

{

cout << "Exception : " << endl;

}

}

// This functions listens for a client

void Server::Listen()

{

try

{

if (listen(serverSocket, 1) == SOCKET\_ERROR)

throw "listen(): Error listening on socket ", WSAGetLastError();

else

cout << "listen() is OK, I'm waiting for connections..." << endl;

}

catch (char \* error) // Catches char error

{

cout << "Exception : " << error << endl;

}

catch (...) // Default catch

{

cout << "Exception : " << endl;

}

}

// This functions checks to see if the server failed to connect

void Server::FailedConnection()

{

fromlen = sizeof(socket\_type);

retval = getsockopt(serverSocket, SOL\_SOCKET, SO\_TYPE, (char \*)&socket\_type, &fromlen);

fromlen = sizeof(from);

try

{

acceptSocket = accept(serverSocket, (SOCKADDR \*)&from, &fromlen);

if (acceptSocket == INVALID\_SOCKET)

{

throw "accept failed: ", WSAGetLastError();

WSACleanup();

}

}

catch (char \* error) // Catches char error

{

cout << "Exception : " << error << endl;

}

catch (...) // Default catch

{

cout << "Exception : " << endl;

}

}

// This function permits an incoming connection on a socket

void Server::AcceptConnection()

{

retval = getnameinfo((SOCKADDR \*)&from,

fromlen,

hoststr,

NI\_MAXHOST,

servstr,

NI\_MAXSERV,

NI\_NUMERICHOST | NI\_NUMERICSERV);

try

{

if (retval != 0)

{

throw "getnameinfo failed: ", retval;

WSACleanup();

}

}

catch (char \* error) // Catches char error

{

cout << "Exception : " << error << endl;

}

catch (...) // Default catch

{

cout << "Exception : " << endl;

}

cout << "Accepted connection from host " << hoststr << " and port " << servstr << endl;

cout << "Accepted connection" << endl;

}

void Server::Receive()

{

while (INFINITE)

{

char receiveBuffer[200] = "";

int byteCount = recv(acceptSocket, receiveBuffer, 200, 0);

try

{

if (byteCount < 0)

{

throw "Client: error %ld.\n", WSAGetLastError();

}

else

{

printf("Received data : %s \n", receiveBuffer);

}

}

catch (char \* error) // Catches char error

{

cout << "Exception : " << error << endl;

}

catch (...) // Default catch

{

cout << "Exception : " << endl;

}

try

{

char Recieved[200] = "Message Recieved";

byteCount = send(acceptSocket, Recieved, 200, 0);

if (byteCount == SOCKET\_ERROR)

{

// Throws error to ServerSendError Class

throw ServerSendError(), WSAGetLastError();

}

else

{

printf("Server: sent %ld bytes \n", byteCount);

}

}

catch (ServerSendError& e) // Catches class error

{

cout << "Error: " << e.what() << endl;

}

catch (...) // Default catch

{

cout << "Error: " << endl;

}

}

}

## **Client.h**

#pragma once

#include "Comms.h"

// Uses Comms as a base class

class Client : public Comms

{

public:

// Initializes server functions

Client();

void Socket();

void Connect();

void Send();

};

## **Client.cpp**

#include "stdafx.h"

#include "Client.h"

Client::Client()

{

}

// This function creates a socket that is bound to a specific transport service provider

void Client::Socket()

{

clientSocket = INVALID\_SOCKET;

clientSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

try

{

if (clientSocket == INVALID\_SOCKET)

{

// Throws error to SocketError Class

throw SocketError(), WSAGetLastError();

WSACleanup();

}

else

{

cout << "socket() is OK!" << endl;

}

}

catch (SocketError& s) // Catches class error

{

cout << "Error: " << s.what() << endl;

}

catch (...) // Default catch

{

cout << "Error: " << endl;

}

}

// This function connects a client to a server(invoked from within the client)

void Client::Connect()

{

clientService.sin\_family = AF\_INET;

InetPton(AF\_INET, \_T("127.0.0.1"), &clientService.sin\_addr.s\_addr);

clientService.sin\_port = htons(port);

try

{

if (connect(clientSocket, (SOCKADDR\*)&clientService, sizeof(clientService)) == SOCKET\_ERROR)

{

throw "Client: connect() - Failed to connect.";

WSACleanup();

}

else

{

cout << "Client: connect() is OK." << endl;

cout << "Client: Can start sending and receiving data..." << endl;

}

}

catch (char \* error) // Catches char error

{

cout << "Exception : " << error << endl;

}

catch (...) // Default catch

{

cout << "Exception : " << endl;

}

}

void Client::Send()

{

while (INFINITE)

{

printf("Enter your message ");

cin.getline(buffer, 200);

int byteCount = send(clientSocket, buffer, 200, 0);

try

{

if (strcmp(buffer, "SHUTDOWN") == 0)

{

abort();

WSACleanup();

}

else if (byteCount == SOCKET\_ERROR)

{

// Throws error to ServerSendError Class

throw ServerSendError(), WSAGetLastError();

}

else

{

printf("Server: sent %ld bytes \n", byteCount);

}

}

catch (ServerSendError& e) // Catches class error

{

cout << "Error: " << e.what() << endl;

}

catch (...) // Default catch

{

cout << "Error: " << endl;

}

char receiveBuffer[200] = "";

byteCount = recv(clientSocket, receiveBuffer, 200, 0);

try

{

if (byteCount < 0)

{

throw "Client send error %ld.\n", WSAGetLastError();

}

else

{

printf("Server: sent %s \n", receiveBuffer);

}

}

catch (char \* error) // Catches char error

{

cout << "Exception : " << error << endl;

}

catch (...) // Default catch

{

cout << "Exception : " << endl;

}

}

}