

Оглавление

[Введение 5](#_Toc168308423)

[1. Постановка задачи 6](#_Toc168308424)

[2. Выбор решения 7](#_Toc168308425)

[3. Протокол взаимодействия 8](#_Toc168308426)

[4. Архитектура сервера 9](#_Toc168308427)

[5. Архитектура клиента 10](#_Toc168308428)

[6. Описание работы программы 11](#_Toc168308429)

[Литература 14](#_Toc168308430)

[Приложение А. Листинг программы 15](#_Toc168308431)

[Приложение А.1. Файл «Main.java». 15](#_Toc168308432)

[Приложение А.2. Файл «Server.java». 15](#_Toc168308433)

[Приложение А.3. Файл «ServerThread.java». 16](#_Toc168308434)

[Приложение А.4. Файл «ServerListener.java». 18](#_Toc168308435)

[Приложение А.5. Файл «Client.java». 20](#_Toc168308436)

[Приложение А.6. Файл «Area.java». 22](#_Toc168308437)

[Приложение А.7. Файл «Tank.java». 26](#_Toc168308438)

[Приложение Б. UML-диаграммы 28](#_Toc168308439)

[Приложение Б.1. UML – диаграмма вариантов использования. 28](#_Toc168308440)

[Приложение Б.2. UML – диаграмма классов. 29](#_Toc168308441)

[Приложение Б.3. UML – диаграмма деятельности. 30](#_Toc168308442)

[Приложение Б.4. UML – диаграмма развертывания. 31](#_Toc168308443)

[Приложение Б.5. UML – диаграмма последовательности. 32](#_Toc168308444)

# Введение

«Клиент - сервер» — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер — это программное обеспечение. Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине. Программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных или в виде сервисных функций (например, работа с электронной почтой, общение посредством систем мгновенного обмена сообщениями или просмотр web-страниц во всемирной паутине). Поскольку одна программа-сервер может выполнять запросы от множества программ-клиентов, её размещают на специально выделенной вычислительной машине, настроенной особым образом, как правило, совместно с другими программами-серверами, поэтому производительность этой машины должна быть высокой. Из-за особой роли такой машины в сети, специфики её оборудования и программного обеспечения, её также называют сервером, а машины, выполняющие клиентские программы, соответственно, клиентами.

# Постановка задачи

Необходимо разработать систему программ клиент-серверной архитектуры, позволяющую пользователю играть, получать информацию с сервера и взаимодействовать с ней.

Функции клиента:

* Обеспечение интерфейса с пользователем
* Взаимодействие с сервером
* Регистрация на сервере

Функции сервера:

* Управление трафиком данных
* Взаимодействие с программами-клиентами

Приложение должно обладать графическим интерфейсом и использовать следующие технологии:

1) Java Collections Framework

2) Механизм обработки исключительных ситуаций

3) Java Stream API

4) Java Multithreading

5) Сетевое взаимодействие.

ОС – Windows. Язык программирования: Java. Среда разработки NetBeans.

# Выбор решения

Важной частью курсовой работы является передача сообщений от клиента к серверу. Для реализации этого используются TCP-сокеты. TCP гарантирует доставку пакетов, их очередность, автоматически разбивает данные на пакеты и контролирует их передачу, в отличии от UDP. Протокол TCP был выбран так как нам необходимо гарантировать доставку данных, UDP.

Для взаимодействия между компьютерами используются адреса и порты. Адрес представляет собой 32-битную структуру. Номер порта – целое число от 0 до 65535.

# Протокол взаимодействия

Разработанная программа основывается на вышеописанных протоколах и состоит из клиента и сервера.

Запускаем сервер и создаем сокет. Устанавливаем локальную точку для прослушивания подключений. Запускаем прослушивание входящих подключений и принимаем новых клиентов и запросы от них в бесконечном цикле. После того, как клиент или запрос был принят, другой запрос отправляется обратно клиенту. На этом установление соединения между клиентом и сервером заканчивается. После сервер в бесконечном цикле ожидает других команд от клиента.

Диаграмма последовательности находиться в Приложение Б.5.

# Архитектура сервера

В курсовой работе для организации взаимодействия сервера с клиентами используется следующий подход: сервер начинает ждать ответ от каждого клиента.

При поступлении запросов от клиентов, он меняет данные в словаре с подключенными клиентами, а также пересылает эти данные другому клиенту.

Диаграмма классов сервера находиться в Приложение Б.2.

# Архитектура клиента

После запуска клиента, он сразу же пытается установить соединение с сервером. После того, как соединение было установлено, пользователь может начинать игровую сессию со своим противником.

Диаграмма классов клиента находиться в Приложение Б.2.

# Описание работы программы

Ниже представлена работа клиент-серверного приложения.

Для начала необходимо запустить сервер:

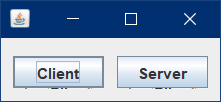


Рисунок 1 - Сервер запущен

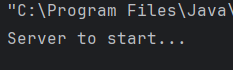


Рисунок 2 - Сервер запущен

Запускаем клиента(-ов). Открывается окно с игровым полем:

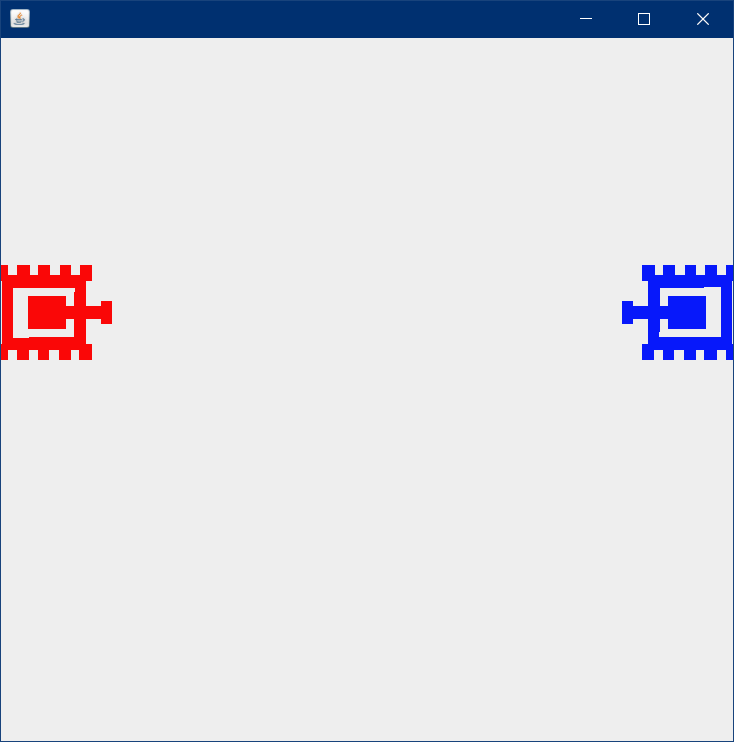


Рисунок 3 – Игровое поле

Если вы поражаете своего противника, программа информирует вас об этом:

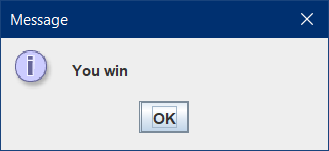


Рисунок 4 – Информационное окно (Победа)

Если поразили вас, то информация об этом так же появляется у вас на экране:

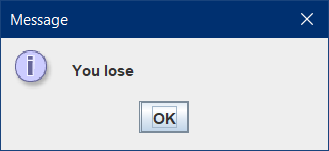


Рисунок 5 - Информационное окно (Поражение)

**Заключение**

В ходе выполнения курсового проектирования были изучены принципы работы протоколов TCP/IP. Были получены навыки разработки приложения с пользовательским интерфейсом.

В итоге было разработано клиент-серверное приложение для передачи текстовых сообщений.

# Литература

1. "Head First Java, Изучаем Java", Кэти Сьерра, Берт Бэйтс
2. "Java. Руководство для начинающих", Герберт Шилдт
3. "Java для чайников", Барри Бёрд
4. "Java. Полное руководство", Герберт Шилдт

# Приложение А. Листинг программы

# Приложение А.1. Файл «Main.java».

import javax.swing.\*;

public class Main {

public static void main(String[] args) {

Form from = new Form();

from.setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

from.pack();

from.setVisible(true);

}

}

# Приложение А.2. Файл «Server.java».

package Server;

import Client.Tank;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.net.Socket;

import java.util.\*;

public class Server

{

private int clientNum;

private Map<Integer, Tank> clients = new HashMap<Integer, Tank>();

private ArrayList<DataOutputStream> listOut = new ArrayList<DataOutputStream>();

private ArrayList<DataInputStream> listIn = new ArrayList<DataInputStream>();

private ArrayList<Socket> listSocket = new ArrayList<Socket>();

// Запуск прослушивания подключения новых игроков

public Server()

{

new ServerThread(this).start();

}

public void setClientNum(int clientNum) { this.clientNum = clientNum; }

public Map<Integer, Tank> getClientInfo() { return clients; }

public int getClientNum() { return clientNum; }

public ArrayList<DataOutputStream> getListOut() { return listOut; }

public ArrayList<DataInputStream> getListIn() { return listIn; }

public ArrayList<Socket> getListSocket() { return listSocket; }

}

# Приложение А.3. Файл «ServerThread.java».

package Server;

import Client.Tank;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.ServerSocket;

public class ServerThread extends Thread {

private Server srv;

public ServerThread(Server srv)

{

super("ServerThread");

this.srv = srv;

}

public void run()

{

// Запуск серверного сокета

try (ServerSocket serverSocket = new ServerSocket(80))

{

System.out.println("Server to start...");

// Проверка канала

while (!serverSocket.isClosed() && srv.getClientNum() < 4)

{

// Ожидание подключения клиента

srv.getListSocket().add(serverSocket.accept());

srv.getListOut().add(new DataOutputStream(srv.getListSocket().getLast().getOutputStream()));

srv.getListIn().add(new DataInputStream(srv.getListSocket().getLast().getInputStream()));

if (srv.getClientNum() == 0) {

srv.getListOut().getLast().writeUTF("Red");

Thread.sleep(10);

synchronized (srv.getClientInfo()) {

srv.getClientInfo().put(srv.getClientNum(), new Tank(0, 200, "src/Assets/tank100RedRight.png", "right"));

}

} else {

srv.getListOut().getLast().writeUTF("Blue");

srv.getListOut().getLast().flush();

Thread.sleep(10);

synchronized (srv.getClientInfo()) {

srv.getClientInfo().put(srv.getClientNum(), new Tank(500, 200, "src/Assets/tank100BlueLeft.png", "left"));

}

}

srv.setClientNum(srv.getClientNum() + 1);

new ServerListener(srv.getListSocket().getLast(), srv.getClientInfo(), srv.getListSocket()).start();

}

}

catch (IOException ex)

{

ex.printStackTrace();

}

catch (InterruptedException e) {

throw new RuntimeException(e);

}

}

}

# Приложение А.4. Файл «ServerListener.java».

package Server;

import Client.Tank;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.Socket;

import java.util.ArrayList;

import java.util.Map;

public class ServerListener extends Thread {

Socket socket;

DataInputStream in;

DataOutputStream out;

Map<Integer, Tank> clients;

ArrayList<Socket> sockets;

public ServerListener(Socket socket, Map<Integer, Tank> clients, ArrayList<Socket> sockets) {

super("ServerListener");

this.socket = socket;

this.clients = clients;

this.sockets = sockets;

}

public void run() {

try {

in = new DataInputStream(socket.getInputStream());

out = new DataOutputStream(socket.getOutputStream());

// Проверяем живой ли канал и работаем если живой

while (!socket.isOutputShutdown()) {

// Проверка на приход от клиента данных

while (in.available() == 0)

Thread.sleep(1);

String data = in.readUTF();

String[] arrStr = data.split(" ");

// Изменение данных об объектах

for (Integer key : clients.keySet()) {

if (key == Integer.parseInt(arrStr[0])) {

synchronized (clients.get(key)) {

clients.get(key).setX(Integer.parseInt(arrStr[1]));

clients.get(key).setY(Integer.parseInt(arrStr[2]));

clients.get(key).setFilePath(arrStr[3]);

clients.get(key).setDir(arrStr[4]);

clients.get(key).setWin(arrStr[5]);

}

}

}

// Отправка измененных данных другому клиенту

for (var item : sockets) {

if (item != socket) {

out = new DataOutputStream(item.getOutputStream());

out.writeUTF(data);

out.flush();

Thread.sleep(10);

}

}

}

}

catch (IOException | InterruptedException ex) {

ex.printStackTrace();

}

}

}

# Приложение А.5. Файл «Client.java».

package Client;

import java.io.\*;

import java.net.Socket;

public class Client {

private Socket socket;

private DataOutputStream oos;

private DataInputStream ois;

public Area area;

private String colorTank;

public Client() {

try {

socket = new Socket("localhost", 80);

ois = new DataInputStream(socket.getInputStream());

oos = new DataOutputStream(socket.getOutputStream());

if (!socket.isOutputShutdown()) {

while (ois.available() == 0)

Thread.sleep(1);

colorTank = ois.readUTF();

}

area = new Area(colorTank);

area.toFront();

new ClientListener().start();

new ClientSender().start();

}

catch (IOException ex) {

ex.printStackTrace();

} catch (InterruptedException e) {

throw new RuntimeException(e);

}

}

// Нить на отправку данных

private class ClientSender extends Thread {

public ClientSender() {

super("ClientSender");

}

public void run() {

Integer numClient;

// Проверяем живой ли канал и работаем если живой

while (!socket.isOutputShutdown()) {

// Проверка на совершение игроком действия

while(!area.getEvent()) {

try {

Thread.sleep(1);

} catch (InterruptedException e) {

throw new RuntimeException(e);

}

}

// Отправка данных на сервер

try {

if (area.getColor().equals("Red"))

numClient = 0;

else

numClient = 1;

oos.writeUTF(numClient.toString() + " " +

area.getTankRed().getX().toString() + " " +

area.getTankRed().getY().toString() + " " +

area.getPath() + " " +

area.getTankRed().getDir() + " " +

area.getTankRed().getWin());

oos.flush();

Thread.sleep(10);

} catch (IOException | InterruptedException e) {

throw new RuntimeException(e);

}

area.setEvent(false);

}

}

}

// Нить прослушивания

private class ClientListener extends Thread {

public ClientListener() {

super("ClientListener");

}

public void run() {

try {

// Проверяем живой ли канал и работаем если живой

while (!socket.isOutputShutdown()) {

// Проверка на приход от сервера данных

while (ois.available() == 0)

Thread.sleep(1);

String data = ois.readUTF();

String[] arrStr = data.split(" ");

// Изменение данных об объектах

area.getTankBlue().setX(Integer.parseInt(arrStr[1]));

area.getTankBlue().setY(Integer.parseInt(arrStr[2]));

area.getTankBlue().setFilePath(arrStr[3]);

area.getTankBlue().setDir(arrStr[4]);

area.getTankBlue().setWin(arrStr[5]);

area.repaint();

}

} catch (IOException | InterruptedException ex) {

ex.printStackTrace();

}

}

}

}

# Приложение А.6. Файл «Area.java».

package Client;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

public class Area extends JFrame implements KeyListener {

private String colorTank;

private Graphics graphics;

private Tank tankRed;

private Tank tankBlue = new Tank(500, 200, "src/Assets/tank100BlueLeft.png", "left");

private boolean isLeft = false;

private boolean isRight = false;

private boolean isUp = false;

private boolean isDown = false;

private boolean event = false;

private String filePath = new String();

public boolean getEvent() { return event; }

public void setEvent(boolean e) { event = e; }

public Tank getTankRed() { return tankRed; }

public Tank getTankBlue() { return tankBlue; }

public String getColor() { return colorTank; }

public String getPath() { return filePath; }

public Area(String colorTank) {

super();

// Определение положение окна на экране

setLocation(200, 100);

// Определение размера окна

setSize (600, 600);

// Открываем окно

setVisible(true);

this.colorTank = colorTank;

if (colorTank.equals("Red")) {

filePath = "src/Assets/tank100" + colorTank + "Right.png";

tankRed = new Tank(0, 200, "src/Assets/tank100" + colorTank + "Right.png", "right");

tankBlue = new Tank(500, 200, "src/Assets/tank100BlueLeft.png", "left");

}

else {

filePath = "src/Assets/tank100" + colorTank + "Left.png";

tankRed = new Tank(500, 200, "src/Assets/tank100" + colorTank + "Left.png", "left");

tankBlue = new Tank(0, 200, "src/Assets/tank100RedRight.png", "right");

}

this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

this.addKeyListener(this);

new MoveThread(this).start();

}

@Override

public void keyPressed(KeyEvent e) {

if (e.getKeyCode()==KeyEvent.VK\_LEFT) {

isLeft = true;

filePath = "src/Assets/tank100" + colorTank + "Left.png";

event = true;

}

if (e.getKeyCode()==KeyEvent.VK\_RIGHT) {

isRight = true;

filePath = "src/Assets/tank100" + colorTank + "Right.png";

event = true;

}

if (e.getKeyCode()==KeyEvent.VK\_UP) {

isUp = true;

filePath = "src/Assets/tank100" + colorTank + "Up.png";

event = true;

}

if (e.getKeyCode()==KeyEvent.VK\_DOWN) {

isDown = true;

filePath = "src/Assets/tank100" + colorTank + "Down.png";

event = true;

}

if (e.getKeyCode()==KeyEvent.VK\_SPACE) {

FireInfo(tankRed.Fire(), tankRed.getDir());

event = true;

}

}

@Override

public void keyReleased(KeyEvent e) {}

private void FireInfo(Integer[] pos, String dir) {

switch (dir) {

case "up":

for (int i = pos[1]; i >= 0; i--) {

if ((tankBlue.getY() <= i && i <= tankBlue.getY() + 100) &&

(tankBlue.getX() <= pos[0] && pos[0] <= tankBlue.getX() + 100))

{

repaint();

tankRed.setWin("w");

break;

}

}

break;

case "down":

for (int i = pos[1]; i < this.getHeight(); i++) {

if ((tankBlue.getY() <= i && i <= tankBlue.getY() + 100) &&

(tankBlue.getX() <= pos[0] && pos[0] <= tankBlue.getX() + 100))

{

repaint();

tankRed.setWin("w");

break;

}

}

break;

case "right":

for (int i = pos[0]; i < this.getWidth(); i++) {

if ((tankBlue.getX() <= i && i <= tankBlue.getX() + 100) &&

(tankBlue.getY() <= pos[1] && pos[1] <= tankBlue.getY() + 100))

{

repaint();

tankRed.setWin("w");

break;

}

}

break;

case "left":

for (int i = pos[0]; i >= 0; i--) {

if ((tankBlue.getX() <= i && i <= tankBlue.getX() + 100) &&

(tankBlue.getY() <= pos[1] && pos[1] <= tankBlue.getY() + 100))

{

repaint();

tankRed.setWin("w");

break;

}

}

break;

}

}

@Override

public void keyTyped(KeyEvent arg0) {}

@Override

public void paint(Graphics g)

{

super.paint(g);

if (tankRed.getWin().equals("w")) {

JOptionPane.showMessageDialog(this, "You win");

tankRed.setWin("l");

}

if (tankBlue.getWin().equals("w")){

JOptionPane.showMessageDialog(this, "You lose");

tankBlue.setWin("l");

event = true;

}

g.drawImage(new ImageIcon(filePath).getImage(), tankRed.getX(), tankRed.getY(), null);

g.drawImage(new ImageIcon(tankBlue.getPath()).getImage(), tankBlue.getX(), tankBlue.getY(), null);

}

private void animate() {

if (isLeft) {

isLeft = false;

tankRed.Left();

this.repaint();

}

if (isRight) {

isRight = false;

tankRed.Right();

this.repaint();

}

if (isUp) {

isUp = false;

tankRed.Up();

this.repaint();

}

if (isDown) {

isDown = false;

tankRed.Down();

this.repaint();

}

}

private class MoveThread extends Thread{

Area runKeyboard;

public MoveThread(Area runKeyboard) {

super("MoveThread");

this.runKeyboard = runKeyboard;

}

public void run(){

while(true) {

runKeyboard.animate();

try {

Thread.sleep(2);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

}

# Приложение А.7. Файл «Tank.java».

package Client;

public class Tank {

private String filePath;

private Integer[] position = new Integer[2];

private String direction;

private String color;

private String win;

public Tank(int x, int y, String path, String dir) {

// Инициализация точек для танка

position[0] = x;

position[1] = y;

filePath = path;

direction = dir;

win = "l";

}

public void setFilePath(String path) { filePath = path; }

public Integer getX() { return position[0]; }

public Integer getY() { return position[1]; }

public void setX(Integer num) { position[0] = num; }

public void setY(Integer num) { position[1] = num; }

public String getDir() { return direction; }

public void setDir(String dir) { direction = dir; }

public void setWin(String w) { win = w; }

public String getPath() { return filePath; }

public String getWin() { return win; }

// Движение вверх

public void Up() {

position[1] -= 20;

direction = "up";

}

// Движение вниз

public void Down() {

position[1] += 20;

direction = "down";

}

// Движение влево

public void Left() {

position[0] -= 20;

direction = "left";

}

// Движение вправо

public void Right() {

position[0] += 20;

direction = "right";

}

// Выстрел

public Integer[] Fire() {

Integer[] pos = new Integer[2];

switch (direction) {

case "up":

pos[0] = position[0] + 50;

pos[1] = position[1];

break;

case "down":

pos[0] = position[0] + 50;

pos[1] = position[1] + 100;

break;

case "left":

pos[0] = position[0];

pos[1] = position[1] + 50;

break;

case "right":

pos[0] = position[0] + 100;

pos[1] = position[1] + 50;

break;

default: break;

}

return pos;

}

}

# Приложение Б. UML-диаграммы

# Приложение Б.1. UML – диаграмма вариантов использования.

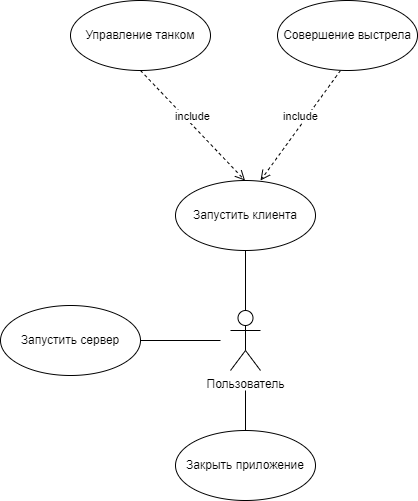


Рисунок 6 - UML- диаграмма вариантов использования

# Приложение Б.2. UML – диаграмма классов.

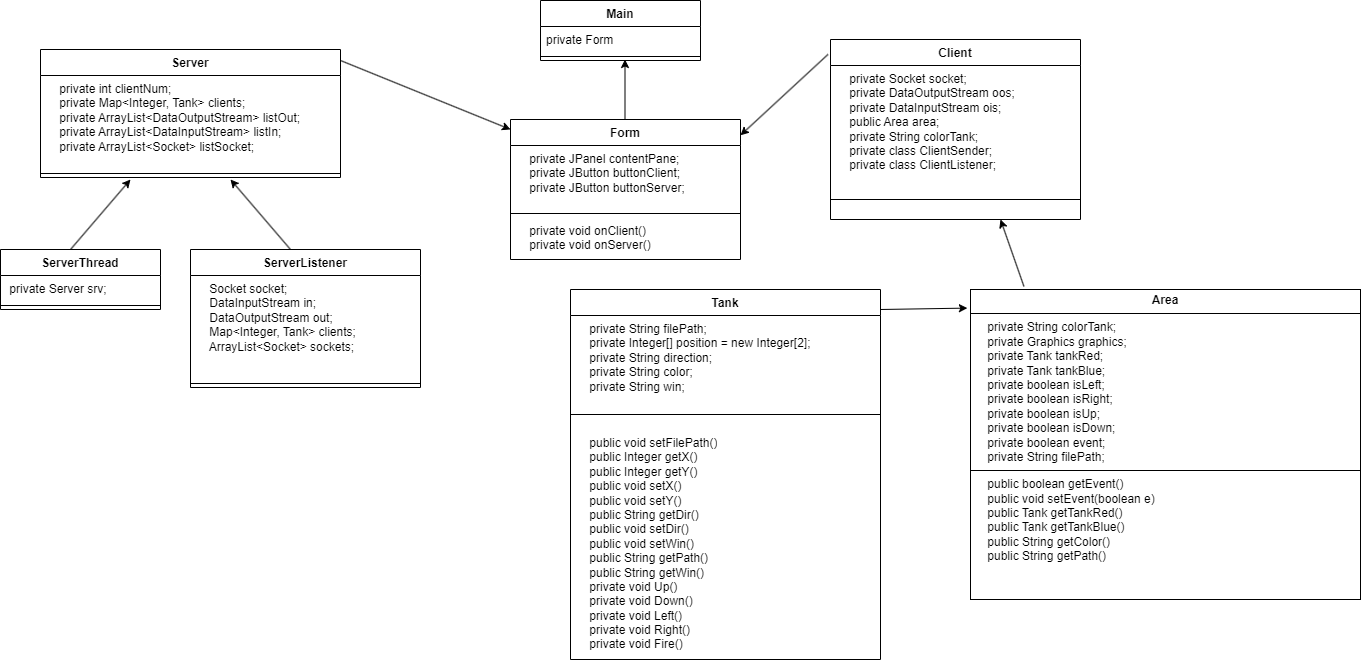


Рисунок 7 - UML- диаграмма класса

# Приложение Б.3. UML – диаграмма деятельности.

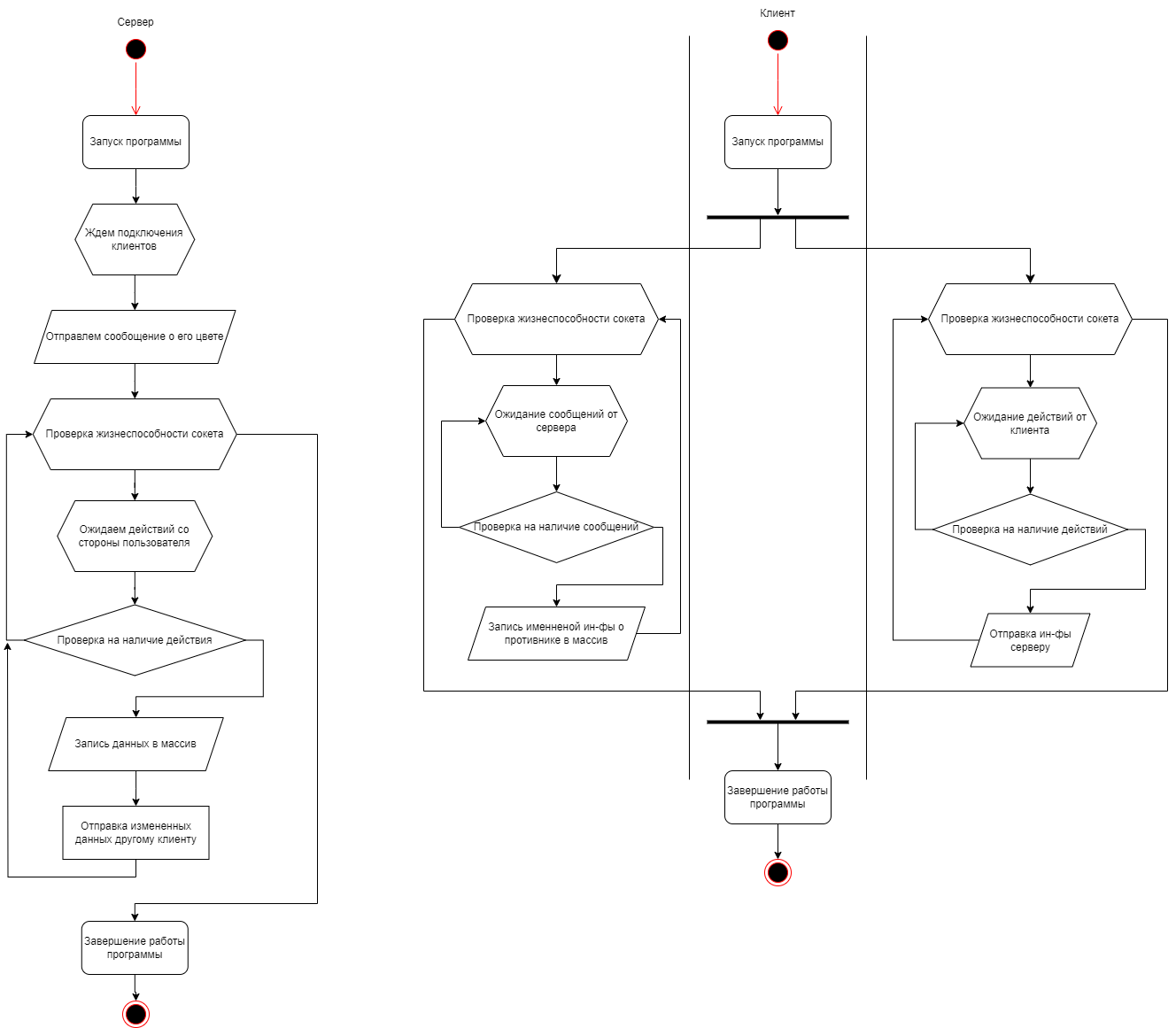


Рисунок 8 - UML- диаграмма деятельности

# Приложение Б.4. UML – диаграмма развертывания.

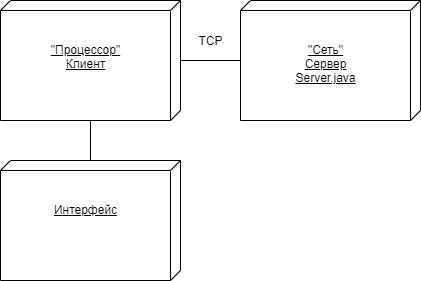


Рисунок 9 - UML- диаграмма развертывания

# Приложение Б.5. UML – диаграмма последовательности.

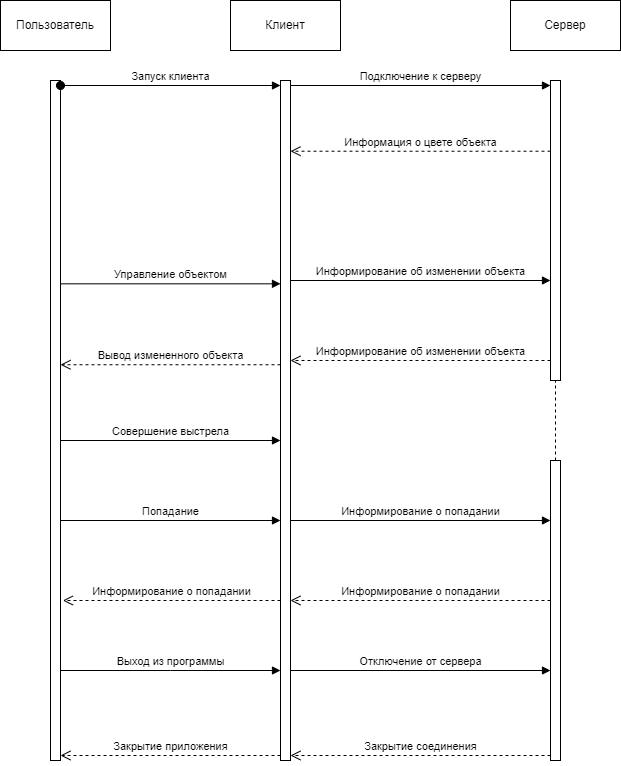


Рисунок 10 - UML- диаграмма последовательности