

Содержание

[Введение 5](#_Toc168559168)

[1 Постановка задачи 7](#_Toc168559169)

[2 Теоретическая часть задания 8](#_Toc168559170)

[2.1 Система программирования 8](#_Toc168559171)

[2.2 Приложение “Сундук” 8](#_Toc168559172)

[3 Описание алгоритма работы 9](#_Toc168559173)

[3.1 Описание клиент-серверной части 9](#_Toc168559174)

[3.2 Описание работы приложения 9](#_Toc168559175)

[4 Описание программы 10](#_Toc168559176)

[4.1 Файл CardGameServer.java 10](#_Toc168559177)

[4.2 Файл CardGameClient.java 10](#_Toc168559178)

[5 Руководство пользователя 11](#_Toc168559179)

[5.1 Сервер 11](#_Toc168559180)

[5.2 Клиент 11](#_Toc168559181)

[Заключение 13](#_Toc168559182)

[Список литературы 14](#_Toc168559183)

[Приложение A. Листинг программы 15](#_Toc168559184)

[Приложение A.1. Файл « CardGameServer.java » 15](#_Toc168559185)

[Приложение A.2. Файл « CardGameClient.java » 18](#_Toc168559186)

[Приложение Б. UML-диаграммы 21](#_Toc168559187)

[Приложение Б.4. UML-диаграмма вариантов использования 21](#_Toc168559188)

[Приложение Б.1. UML-диаграмма классов 22](#_Toc168559189)

[Приложение Б.3. UML- диаграмма деятельности 24](#_Toc168559190)

[Приложение Б.5. UML-диаграмма последовательности 25](#_Toc168559191)

[Приложение Б.2. UML-диаграмма развертывания 26](#_Toc168559192)

# Введение

На сегодняшний момент язык Java является одним из самых распространенных и популярных языков программирования.

Ключевой особенностью языка Java является то, что его код сначала транслируется в специальный байт-код, независимый от платформы. А затем этот байт-код выполняется виртуальной машиной JVM (Java Virtual Machine).

Подобная архитектура обеспечивает кроссплатформенность и аппаратную переносимость программ на Java, благодаря чему подобные программы без перекомпиляции могут выполняться на различных платформах - Windows, Linux, Mac OS и т.д. Для каждой из платформ может быть своя реализация виртуальной машины JVM, но каждая из них может выполнять один и тот же код.

Java является языком с Си-подобным синтаксисом и близок в этом отношении к C/C++ и C#. Поэтому, если вы знакомы с одним из этих языков, то овладеть Java будет легче.

Еще одной ключевой особенностью Java является то, что она поддерживает автоматическую сборку мусора. А это значит, что вам не надо освобождать вручную память от ранее использовавшихся объектов, как в С++, так как сборщик мусора это сделает автоматически за вас.

Java является объектно-ориентированным языком. Он поддерживает полиморфизм, наследование, статическую типизацию. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений.

В данном курсовом проекте будет использована клиент-серверная архитектура. Клиент-сервер (Client-server) – вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг (сервисов), называемыми серверами, и заказчиками услуг, называемыми клиентами.

Нередко клиенты и серверы взаимодействуют через компьютерную сеть и могут быть как различными физическими устройствами, так и программным обеспечением.

Сама технология очень проста. Пользователь делает запрос, а сервер выдает ответ.

# Постановка задачи

Разработать многомодульное приложение “Шахматы”. Язык программирования Java, среда разработки IntelliJ IDEA. Приложение должно обладать графическим интерфейсом и использовать следующие технологии:

1. Java Collection Framework;

2. Механики обработки исключительных ситуаций;

3. Java Stream API;

4. Java Multithreading;

5. Сетевое взаимодействие.

# Теоретическая часть задания

## Система программирования

IntelliJ IDEA — интегрированная среда разработки программного обеспечения для многих языков программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains.

Первая версия появилась в январе 2001 года и быстро приобрела популярность как первая среда для Java с широким набором интегрированных инструментов для рефакторинга[6], которые позволяли программистам быстро реорганизовывать исходные тексты программ. Дизайн среды ориентирован на продуктивность работы программистов, позволяя сконцентрироваться на функциональных задачах, в то время как IntelliJ IDEA берёт на себя выполнение рутинных операций.

## Приложение “Сундук”

Данное приложение реализует карточную игру по сети с другом. Управление осуществляется только мышкой,после выбора карты игроки видят чей ход в данный момент игры. Все необходимые функции были реализованы.

# Описание алгоритма работы

## Описание клиент-серверной части

Разработанная программа состоит из клиента и сервера.

Сначала необходимо запустить сервер с уникальным адресом для обслуживания TCP/IP, определяемый комбинацией IP-адреса хоста с номером порта обслуживания, который создает конечную точку для обслуживания.

Далее сокет привязывается к локальной конечной точке. Чтобы сокет клиента мог идентифицировать потоковый сокет TCP, серверная программа присваивает имя своему сокету.

Затем сокет переходит в состояние прослушивания, в котором он будет ожидать входящие попытки соединения от клиентов. В случаи обнаружения входящего соединения, сервер предоставляет клиенту нить для обслуживания. На этом установление соединения между клиентом и сервером заканчивается.

После установления соединения сервер ожидает запросы от клиентов.

Как только клиенты отправляют запросы, сервер их обрабатывает и присылает ответы на данные запросы.

## Описание работы приложения

Приложение после входа одного из игроков ожидает подключение второго для игры. После подключения игрокам в случайном порядке раздаются 7 семь карт. Затем одному из игроков предоставляется выбор карты, если игрок угадывает номер карты, тогда его ход повторяется, выигрывает тот, кто быстрее избавится от всех карт.

# Описание программы

## Файл CardGameServer.java

Данный файл описывает логику работы сервера и подключение 2 клиентов. Он обрабатывает запрос от клиента и реализует подключение двух пользователей. Здесь содержится описание нити, которая будет создаваться при появлении нового запроса клиента, создание сокета по которому будут подключаться клиенты, выделение нити клиента, обслуживание запросов клиента нитью.

Сервер обслуживает следующие запросы содержащие в себе IP и порт. При успешном подключении приложение запускает игру между двумя игроками, при обрыве соединения приложение завершает работу.

## Файл CardGameClient.java

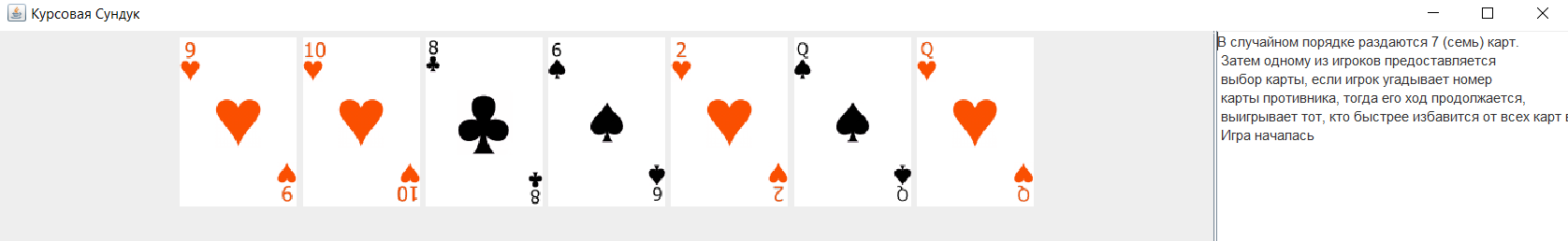
Данный файл описывает логику клиента. Подключение к серверу, получение карт от сервера, при их добавлении, рестарт игры,происходит отображение карт у каждого из пользователей, как указано в компоненте(рис.1). 

Рис.1- Отображение у клиента

# Руководство пользователя

## Сервер

Сервер не имеет графического интерфейса. Требуется только запустить его, дальше он сам будет принимать запросы и обслуживать их.

## Клиент

При запуске клиента, открывается главное меню приложения, в котором ожидается подключение второго клиента (рисунок 2).

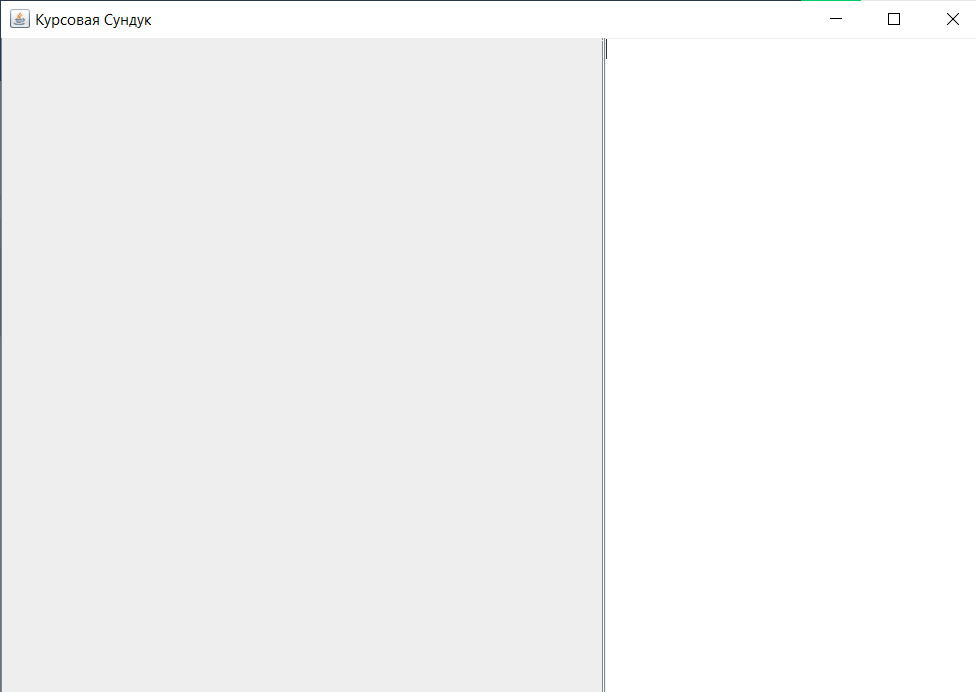


Рис.2 – Ожидание второго клиента

После подключения второго клиента начнется игра (рисунок 3).

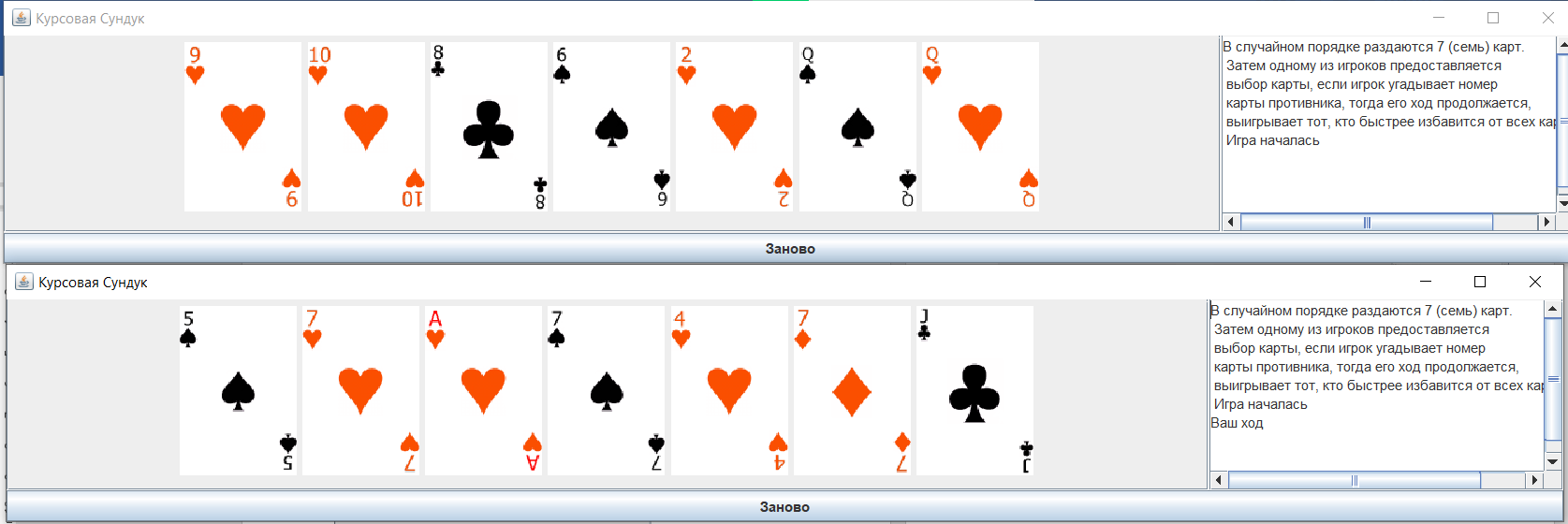


Рис.3 – Начало игры

В случае выигрыша одной из сторон, приложение выдаст соответствующее сообщение и завершит работу (рисунок 4).

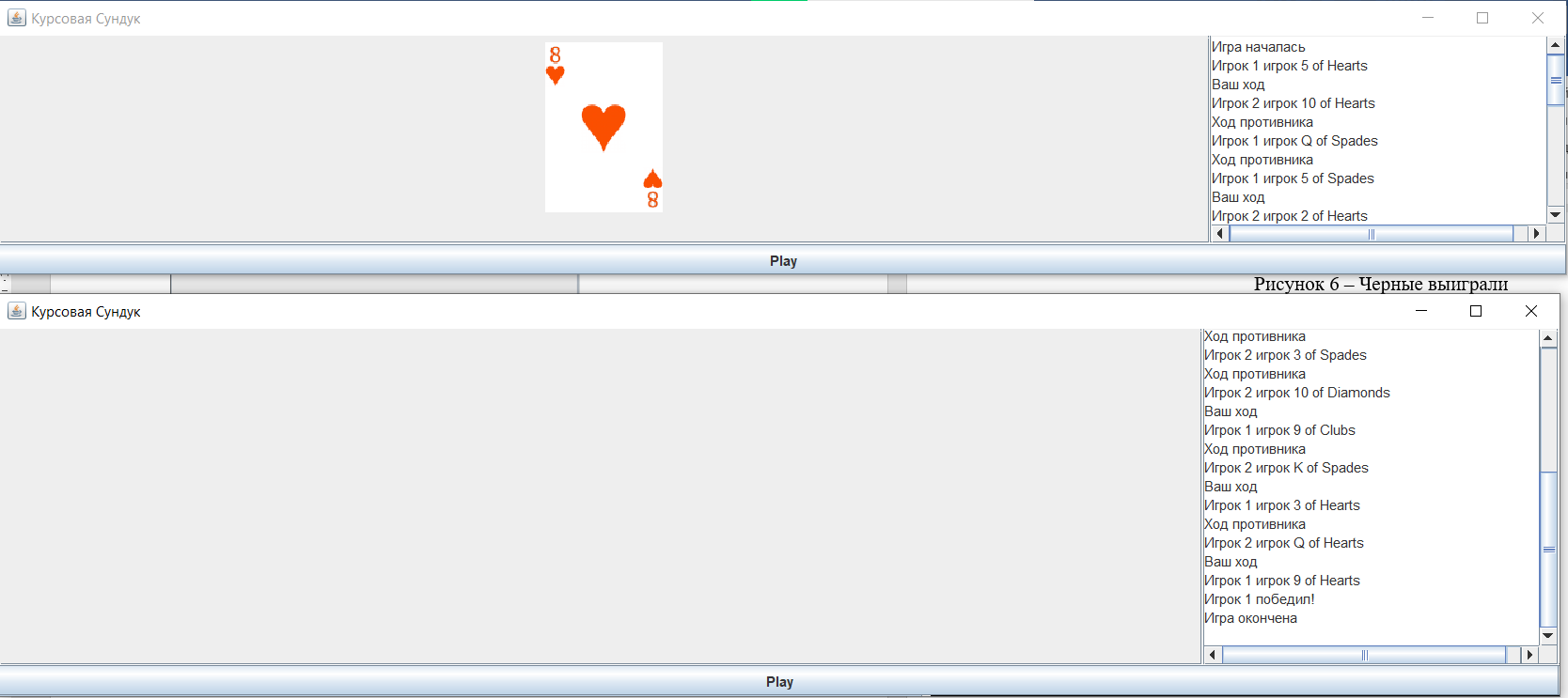


Рисунок 4 – Победа 1 игрока

# Заключение

При выполнении данного курсового проекта были изучены принципы работы протоколов TCP/IP. Были получены навыки разработки многомодульного приложения с пользовательским интерфейсом и работы в среде IntelliJ IDEA. Получены навыки разработки клиент-серверных приложений.

В результате выполнения данной курсовой работы была разработано модульное приложение «Сундук» , позволяющее пользователям играть либо между собой в карты.

# Список литературы

1. IntelliJ IDEA. Профессиональное программирование на Java / С. Давыдов, А. Ефимов. - М.: БХВ-Петербург, 2005. - 800 c. (Дата обращения: 15.05.24) [IntelliJ IDEA. Профессиональное программирование на Java, А. А. Ефимов – скачать pdf на ЛитРес (litres.ru)](https://www.litres.ru/book/aleksey-alekseevich/intellij-idea-professionalnoe-programmirovanie-na-jav-647725/)
2. Язык Java. Курс программирования / Уолтер Савитч. - М.: Вильямс, 2015. - 928 c. (Дата обращения: 15.05.24) [Язык Java. Курс программирования — Савитч, Уолтер | Livelib](https://www.livelib.ru/book/1000444090-yazyk-java-kurs-programmirovaniya-savitch?ysclid=lx2w3nhkfi251021343)
3. "Swing: Эффектные пользовательские интерфейсы", Иван Портянкин — Издательство «Лори», 2011 — 591 с. (Дата обращения: 15.05.24) (Дата обращения: 15.05.24) [Swing. Эффектные пользовательские интерфейсы — Иван Портянкин | Livelib](https://www.livelib.ru/book/1000812065-swing-effektnye-polzovatelskie-interfejsy-ivan-portyankin?ysclid=lx2w472k6v349211673)
4. Форум программистов и сисадминов Киберфорум. <http://www.cyberforum.ru/>.(Дата обращения: 15.05.24)
5. Stack Overflow <https://stackoverflow.com/> . (Дата обращения: 15.05.24)

# Приложение A. Листинг программы

## Приложение A.1. Файл « CardGameServer.java »

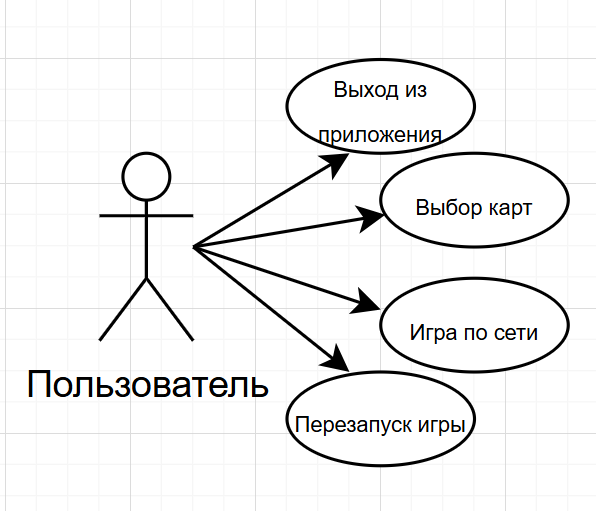
import java.io.\*;  
import java.net.\*;  
import java.util.\*;  
  
public class CardGameServer {  
 private static final int *PORT* = 12345;  
 private static final int *MAX\_PLAYERS* = 2;  
 private static final List<ClientHandler> *players* = new ArrayList<>();  
 private static final List<String> *deck* = new ArrayList<>();  
 private static int *currentPlayerIndex* = 0;  
 private static boolean *gameStarted* = false;  
  
 public static void main(String[] args) {  
 try (ServerSocket serverSocket = new ServerSocket(*PORT*)) {  
 System.*out*.println("Сервер запущен на порту " + *PORT*);  
  
 *initializeDeck*();  
  
 while (*players*.size() < *MAX\_PLAYERS*) {  
 Socket socket = serverSocket.accept();  
 ClientHandler player = new ClientHandler(socket, *players*.size());  
 *players*.add(player);  
 new Thread(player).start();  
 System.*out*.println("Игрок " + *players*.size() + " подключился.");  
 }  
  
 if (*players*.size() == *MAX\_PLAYERS*) {  
 *gameStarted* = true;  
 *startGame*();  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private static void initializeDeck() {  
 String[] suits = {"Hearts", "Diamonds", "Clubs", "Spades"};  
 String[] values = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A"};  
 for (String suit : suits) {  
 for (String value : values) {  
 *deck*.add(value + " of " + suit);  
 }  
 }  
 Collections.*shuffle*(*deck*);  
 }  
  
 private static void startGame() {  
 try {  
 for (ClientHandler player : *players*) {  
 for (int i = 0; i < 7; i++) {  
 player.sendCard(*deck*.remove(0));  
 }  
 player.sendMessage("Игра началась");  
 }  
 *players*.get(*currentPlayerIndex*).sendMessage("Ваш ход");  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private static void checkForWin() {  
 for (ClientHandler player : *players*) {  
 if (player.getHandSize() == 0) {  
 *sendMessageToAll*("Игрок " + (player.getPlayerIndex() + 1) + " победил!");  
 *endGame*();  
 break;  
 }  
 }  
 }  
  
 private static boolean checkHandPlayer(ClientHandler player, String card) {  
 return player.getHand().stream().anyMatch(s -> s.contains(card));  
 }  
  
 private static void sendMessageToAll(String message) {  
 for (ClientHandler player : *players*) {  
 try {  
 player.sendMessage(message);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 private static void endGame() {  
 for (ClientHandler player : *players*) {  
 try {  
 player.sendMessage("Игра окончена");  
 player.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 System.*exit*(0);  
 }  
  
 private static class ClientHandler implements Runnable {  
 private final Socket socket;  
 private final int playerIndex;  
 private final BufferedReader in;  
 private final PrintWriter out;  
 private final List<String> hand;  
  
 public ClientHandler(Socket socket, int playerIndex) throws IOException {  
 this.socket = socket;  
 this.playerIndex = playerIndex;  
 this.in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 this.out = new PrintWriter(socket.getOutputStream(), true);  
 this.hand = new ArrayList<>();  
 }  
  
 @Override  
 public void run() {  
 try {  
 while (true) {  
 String message = in.readLine();  
 if (message != null && message.startsWith("PLAY")) {  
 String[] parts = message.split(" ");  
 String card = parts[1] + " " + parts[2] + " " + parts[3];  
 if (hand.contains(card)) {  
 hand.remove(card);  
 System.*out*.println("Игрок " + (playerIndex + 1) + " игрок " + card);  
 *sendMessageToAll*("Игрок " + (playerIndex + 1) + " игрок " + card);  
 *checkForWin*();  
 if(!*checkHandPlayer*(*players*.get((*currentPlayerIndex* + 1) % *MAX\_PLAYERS*), parts[1])){// || !checkHandPlayer(players.get((currentPlayerIndex + 1) % MAX\_PLAYERS), parts[3])) {  
 *currentPlayerIndex* = (*currentPlayerIndex* + 1) % *MAX\_PLAYERS*;  
 }  
 *players*.get(*currentPlayerIndex*).sendMessage("Ваш ход:");  
 *players*.get((*currentPlayerIndex*+1)%*MAX\_PLAYERS*).sendMessage("Ход противника:");  
 } else {  
 sendMessage("Плохая карта. Рисуем карту из колоды.");  
 if (!*deck*.isEmpty()) {  
 sendCard(*deck*.remove(0));  
 } else {  
 sendMessage("Колода пуста.");  
 }  
 }  
 }  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 } finally {  
 try {  
 socket.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 public void sendCard(String card) throws IOException {  
 hand.add(card);  
 out.println("CARD " + card);  
 }  
  
 public void sendMessage(String message) throws IOException {  
 out.println(message);  
 }  
  
 public int getHandSize() {  
 return hand.size();  
 }  
  
 public List<String> getHand() {  
 return hand;  
 }  
  
 public int getPlayerIndex() {  
 return playerIndex;  
 }  
  
 public void close() throws IOException {  
 socket.close();  
 }  
 }  
}

## Приложение A.2. Файл « CardGameClient.java »

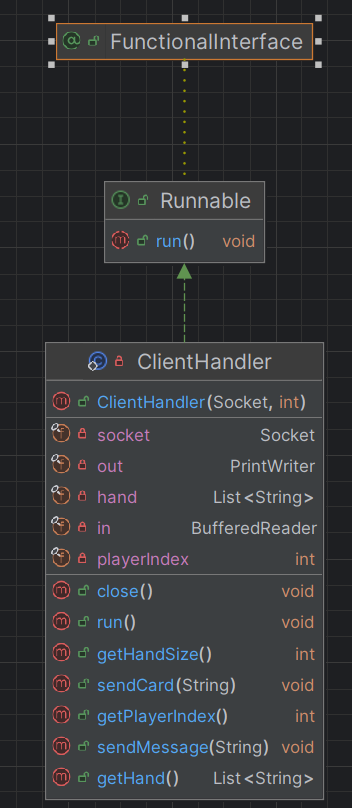
import javax.imageio.ImageIO;  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.image.BufferedImage;  
import java.io.\*;  
import java.net.Socket;  
import java.util.HashMap;  
import java.util.Map;  
  
public class CardGameClient extends JFrame {  
 private static final String *SERVER\_HOST* = "localhost";  
 private static final int *SERVER\_PORT* = 12345;  
  
 private final DefaultListModel<String> handModel;  
 private final JList<String> handList;  
 private final JButton playButton;  
 private final JTextArea messageArea;  
 private final JPanel cardPanel;  
 private BufferedReader in;  
 private PrintWriter out;  
  
 private final Map<String, BufferedImage> cardImages = new HashMap<>();  
 private boolean turn = false; // Переменная для отслеживания хода  
  
 public CardGameClient() {  
 super("Курсовая Сундук");  
  
 handModel = new DefaultListModel<>();  
 handList = new JList<>(handModel);  
 playButton = new JButton("Play");  
 messageArea = new JTextArea(10, 30);  
 messageArea.setEditable(false);  
 cardPanel = new JPanel();  
  
 loadCardImages();  
  
 setLayout(new BorderLayout());  
 add(new JScrollPane(cardPanel), BorderLayout.*CENTER*);  
 add(playButton, BorderLayout.*SOUTH*);  
 add(new JScrollPane(messageArea), BorderLayout.*EAST*);  
  
 playButton.addActionListener(this::playCard);  
  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 setSize(800, 600);  
 setVisible(true);  
  
 connectToServer();  
 }  
  
 private void loadCardImages() {  
 String[] suits = {"Hearts", "Diamonds", "Clubs", "Spades"};  
 String[] values = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A"};  
 for (String suit : suits) {  
 for (String value : values) {  
 try {  
 String cardName = value + " of " + suit;  
 BufferedImage img = ImageIO.*read*(new File("cards/" + cardName + ".png"));  
 cardImages.put(cardName, img);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
  
 private void connectToServer() {  
 try {  
 Socket socket = new Socket(*SERVER\_HOST*, *SERVER\_PORT*);  
 in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 out = new PrintWriter(socket.getOutputStream(), true);  
  
 new Thread(new IncomingReader()).start();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private void playCard(ActionEvent e) {  
 // This method is not used anymore. Cards are played by clicking on their images.  
 }  
  
 private class IncomingReader implements Runnable {  
 @Override  
 public void run() {  
 try {  
 String message;  
 while ((message = in.readLine()) != null) {  
 if (message.startsWith("CARD")) {  
 String card = message.substring(5);  
 handModel.addElement(card);  
 addCardToPanel(card);  
 turn = true;  
 } else if (message.equals("Ваш ход:")) {  
 turn = true; // Устанавливаем, что сейчас ход пользователя  
 messageArea.append("Ваш ход\n");  
 } else if (message.equals("Ход противника:")) {  
 turn = false; // Устанавливаем, что сейчас ход противника  
 messageArea.append("Ход противника\n");  
 } else {  
 messageArea.append(message + "\n");  
 if (message.equals("Игра окончена")) {  
 out.close();  
 in.close();  
 }  
 }  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 private void addCardToPanel(String card) {  
 System.*out*.println("Добавление карты в руку: " + card); // Логирование для отладки  
 BufferedImage img = cardImages.get(card);  
 if (img != null) {  
 ImageIcon icon = new ImageIcon(img);  
 JLabel cardLabel = new JLabel(icon);  
 cardLabel.setToolTipText(card);  
 cardLabel.addMouseListener(new java.awt.event.MouseAdapter() {  
 @Override  
 public void mouseClicked(java.awt.event.MouseEvent evt) {  
 if (turn) { // Проверяем, если это ход пользователя  
 out.println("PLAY " + card);  
 handModel.removeElement(card);  
 cardPanel.remove(cardLabel);  
 cardPanel.revalidate();  
 cardPanel.repaint();  
 turn = false; // После хода пользователя, устанавливаем, что ход окончен  
 }  
 }  
 });  
 cardPanel.add(cardLabel);  
 cardPanel.revalidate();  
 cardPanel.repaint();  
 } else {  
 System.*out*.println("Изображение для карты не найдено: " + card);  
 }  
 }  
  
 public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(CardGameClient::new);  
 }  
}

# Приложение Б. UML-диаграммы

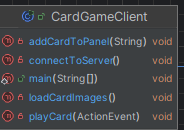
## Приложение Б.4. UML-диаграмма вариантов использования



## Приложение Б.1. UML-диаграмма классов

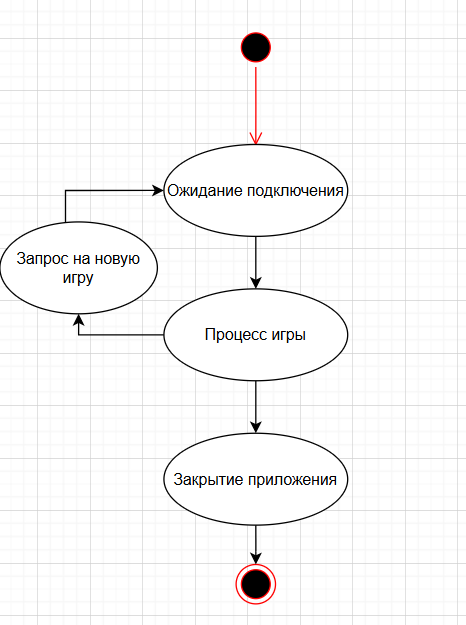


Сервер

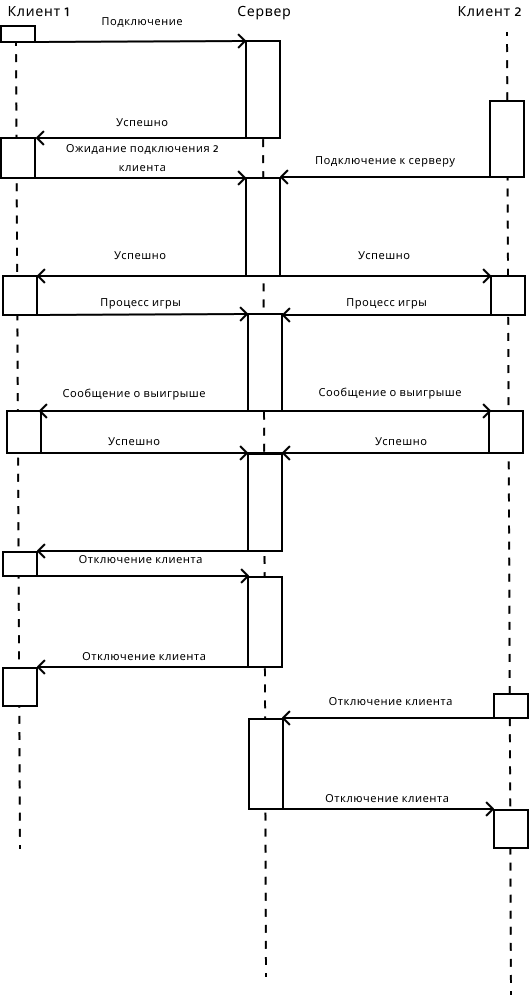


Клиент

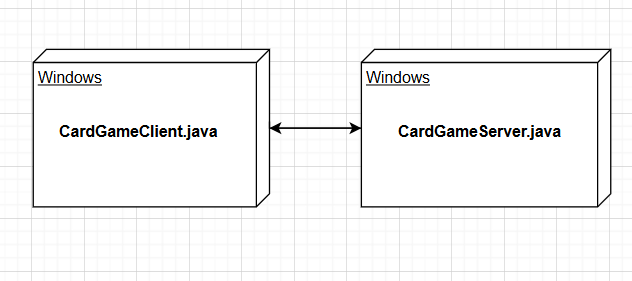
## Приложение Б.3. UML- диаграмма деятельности



## Приложение Б.5. UML-диаграмма последовательности



## Приложение Б.2. UML-диаграмма развертывания

****