Національний Технічний Університет України

Київський Політехнічний Інститут імені Ігоря Сікорського

Інститут Прикладного Системного Аналізу

Кафедра Системного Проектування

Паралельні обчислення

Лабораторна робота №5

Виконала:

Желєзнова В.С.

Група: ДА-81

Перевірив: Яременко В.С.

Київ – 2021

ЗМІСТ

[Мета роботи 3](#_Toc70287781)

[Завдання 3](#_Toc70287782)

[Лістинг програми мовою Java 4](#_Toc70287783)

[Dealer.java – сервер 4](#_Toc70287784)

[Client.java 7](#_Toc70287785)

[Deck.java – колода, рука гравця та дилера 8](#_Toc70287786)

[Card.java – карта 10](#_Toc70287787)

[Suit.java – масть 11](#_Toc70287788)

[Value.java – значення 11](#_Toc70287789)

[Результати роботи програми 12](#_Toc70287790)

[Опис роботи програми 13](#_Toc70287791)

[Висновки 14](#_Toc70287792)

Мета роботи

Дослідження мережевих можливостей розподіленних систем та реалізація клієнт-серверного додатку.

Завдання

Розробити клієнт-серверний додаток, використовуючи Java RMI, Sockets або аналогічну технологію.

**Варіант 10**

Гра по мережі в «21».

Лістинг програми мовою Java

Dealer.java – сервер

import java.io.\*;  
import java.net.\*;  
  
public class Dealer {  
  
 // необходимые переменные для работы сервера  
 private static Socket *clientSocket*;  
 private static ServerSocket *server*;  
 private static BufferedReader *in*;  
 private static BufferedWriter *out*;  
 public static boolean *endGame* = false;  
  
 public static void main(String[] args){  
 try{  
 try{  
 // привязываю серверный сокет к порту  
 *server* = new ServerSocket(3000);  
 System.*out*.println("Welcome to BlackJack!");  
 // прослушиваю подключение клиента к серверному сокету   
 *clientSocket* = *server*.accept();  
 System.*out*.println("Player connected!");  
 try{  
 // необходимые переменные для InputStream и OutputStream  
 *in* = new BufferedReader(new InputStreamReader(*clientSocket*.getInputStream()));  
 *out* = new BufferedWriter(new OutputStreamWriter(*clientSocket*.getOutputStream()));  
  
 // создаю объект колоды, заполняю колоду и перемешиваю карты в ней   
 Deck playingDeck = new Deck();  
 playingDeck.createFullDeck();  
 playingDeck.shuffle();  
 System.*out*.println("Deck is created!");  
  
 // создаю объекты руки игрока и руки дилера  
 Deck playerCards = new Deck();  
 Deck dealerCards = new Deck();  
  
 *out*.write("STARTING THE GAME!\n");  
 *out*.flush();  
  
 while(!*endGame*) {  
 *out*.write("Dealing...\n");  
 *out*.flush();  
  
 // набираю 2 карты в руку игрока   
 playerCards.draw(playingDeck);  
 playerCards.draw(playingDeck);  
  
 // набираю 2 карты в руку дилера   
 dealerCards.draw(playingDeck);  
 dealerCards.draw(playingDeck);  
  
 while (true) {  
 // вывожу руку игрока, значение руки игрока, руку дилера со скрытой картой и вопрос брать еще карту, или пасовать   
 *out*.write("Your Hand: " + playerCards.toString() + "\n");  
 *out*.flush();  
  
 *out*.write("Your hand is currently valued at: " + playerCards.cardsValue() + "\n");  
 *out*.flush();  
  
 *out*.write("Dealer Hand: " + dealerCards.getCard(0).toString() + " and [hidden]\n");  
 *out*.flush();  
  
 *out*.write("Would you like to 1 - Hit or 2 - Stand?\n");  
 *out*.flush();  
  
 // считываю ответ клиента   
 String clientWord = *in*.readLine();  
 System.*out*.println(clientWord);  
  
 if (clientWord.equals("1") && clientWord != null) {  
 // набираю еще одну карту в руку   
 playerCards.draw(playingDeck);  
 *out*.write("You draw a:" + playerCards.getCard(playerCards.deckSize() - 1).toString() + "\n");  
 *out*.flush();  
  
 // в случае, если набралось больше 21, игрок проигрывает  
 if (playerCards.cardsValue() > 21) {  
 *out*.write("YOU LOST. Currently valued at: " + playerCards.cardsValue() + "\n");  
 *out*.flush();  
 *endGame* = true;  
 break;  
 }  
 }  
  
 // останавливаем цикл набора карт, если игрок выбрал пас   
 if (clientWord.equals("2") && clientWord != null) {  
 break;  
 }  
 }  
  
 // выводим карты дилера   
 *out*.write("Dealer Cards:" + dealerCards.toString() + "\n");  
 *out*.flush();  
  
 // если рука дилера больше руки игрока, то дилер побеждает   
 if((dealerCards.cardsValue() > playerCards.cardsValue())&& *endGame* == false){  
 *out*.write("Dealer beats you " + dealerCards.cardsValue() + " to " + playerCards.cardsValue() + "\n");  
 *out*.flush();  
 *endGame* = true;  
 }  
  
 // если у дилера сумма карт меньше 17, то он добирает еще одну карту  
 while((dealerCards.cardsValue() < 17) && *endGame* == false){  
 dealerCards.draw(playingDeck);  
 *out*.write("Dealer draws: " + dealerCards.getCard(dealerCards.deckSize()-1).toString() + "\n");  
 *out*.flush();  
 }  
  
 *out*.write("Dealers hand value: " + dealerCards.cardsValue() + "\n");  
 *out*.flush();  
  
 // если рука дилера больше 21, он проигрывает и выигрывает клиент   
 if((dealerCards.cardsValue()>21)&& *endGame* == false){  
 *out*.write("Dealer LOST. YOU WIN!\n");  
 *out*.flush();  
 *endGame* = true;  
 }  
  
 // если рука дилера равна руке игрока, то ничья  
 if((dealerCards.cardsValue() == playerCards.cardsValue()) && *endGame* == false){  
 *out*.write("Push.\n");  
 *out*.flush();  
 *endGame* = true;  
 }  
  
 // если рука игрока, больше руки дилера, то игрок побеждает  
 if((playerCards.cardsValue() > dealerCards.cardsValue()) && *endGame* == false){  
 *out*.write("You WIN the hand.\n");  
 *out*.flush();  
 *endGame* = true;  
 } else if(*endGame* == false) {  
 *out*.write("Dealer wins.\n");  
 *out*.flush();  
 *endGame* = true;  
 }  
  
 // после окончания игры помещаем все карты из рук дилера и игрока обратно в колоду   
 // (было расчитано на игру с денежными ставками и несколькими раундами)  
 playerCards.moveAllToDeck(playingDeck);  
 dealerCards.moveAllToDeck(playingDeck);  
 *out*.write("End of Hand.\n");  
 *out*.flush();  
 }  
 } finally {  
 // закрываем сокет клиента и InputStream, OutputStream  
 *clientSocket*.close();  
 *in*.close();  
 *out*.close();  
 }  
 } finally {  
 // закрываем сервер   
 System.*out*.println("Server is closed!");  
 *server*.close();  
 }  
 } catch (IOException e) {  
 System.*err*.println(e);  
 }  
 }  
  
 public static boolean isEndGame() {  
 return *endGame*;  
 }  
   
}

Client.java

import java.io.\*;  
import java.net.\*;  
  
public class Client {  
  
 // необходимые переменные для работы клиента  
 private static Socket *clientSocket*;  
 private static BufferedReader *reader*;  
 private static BufferedReader *in*;  
 private static BufferedWriter *out*;  
  
 public static void main(String[] args){  
 try{  
 try{  
 // попытка подключения киента к серверу  
 *clientSocket* = new Socket("localhost", 3000);  
 // необходимые переменные InputStream, OutputStream  
 *reader* = new BufferedReader(new InputStreamReader(System.*in*));  
 *in* = new BufferedReader(new InputStreamReader(*clientSocket*.getInputStream()));  
 *out* = new BufferedWriter(new OutputStreamWriter(*clientSocket*.getOutputStream()));  
 boolean endGame = Dealer.*isEndGame*();  
 String serverWord;  
 serverWord = *in*.readLine(); //starting  
 System.*out*.println(serverWord);  
  
 // заходим в цикл пока не закончится игра  
 while(!endGame){  
 serverWord = *in*.readLine(); //Dealing  
 System.*out*.println(serverWord);  
  
 serverWord = *in*.readLine(); //Your Hand  
 System.*out*.println(serverWord);  
  
 serverWord = *in*.readLine(); //Your Value  
 System.*out*.println(serverWord);  
  
 serverWord = *in*.readLine(); //Dealer Hand  
 System.*out*.println(serverWord);  
  
 serverWord = *in*.readLine(); //Hit or Stand  
 System.*out*.println(serverWord);  
  
 String clientWord = *reader*.readLine(); //  
 *out*.write(clientWord + "\n");  
 *out*.flush();  
  
 if (clientWord.equals("Hit") && clientWord != null){  
 serverWord = *in*.readLine();  
 System.*out*.println(serverWord);  
 serverWord = *in*.readLine();  
 System.*out*.println(serverWord);  
 endGame = Dealer.*isEndGame*();  
 }  
  
 serverWord = *in*.readLine();  
 System.*out*.println(serverWord);  
  
 serverWord = *in*.readLine();  
 System.*out*.println(serverWord);  
 endGame = Dealer.*isEndGame*();  
  
 serverWord = *in*.readLine();  
 System.*out*.println(serverWord);  
  
 serverWord = *in*.readLine();  
 System.*out*.println(serverWord);  
 endGame = Dealer.*isEndGame*();  
  
 serverWord = *in*.readLine();  
 System.*out*.println(serverWord);  
 }  
 } finally {  
 // закрываем клиент и стримы ввода/вывода/считывателя  
 System.*out*.println("Client was closed");  
 *clientSocket*.close();  
 *in*.close();  
 *out*.close();  
 *reader*.close();  
 }  
 } catch (IOException e) {  
 System.*err*.println(e);  
 }  
 }  
}

Deck.java – колода, рука гравця та дилера

import java.util.ArrayList;  
import java.util.Random;  
  
public class Deck {  
  
 private ArrayList<Card> cards;  
  
 public Deck(){  
 //Создание новой колоды игральных карт  
 this.cards = new ArrayList<Card>();  
  
 }  
  
 //Добавляем 52 карты в колоду  
 public void createFullDeck(){  
 //Генерация карт  
 //Цикл по мастям  
 for(Suit cardSuit : Suit.*values*()){  
 //Цикл по значениям  
 for(Value cardValue : Value.*values*()){  
 //Добавление новой карты  
 this.cards.add(new Card(cardSuit,cardValue));  
 }  
 }  
 }  
  
  
 //Перемешиваем колоду карт  
 public void shuffle(){  
 //Создаем новый arraylist для временного хранения перетасованных карт  
 ArrayList<Card> tmpDeck = new ArrayList<Card>();  
 //Рандомно достаем из первоначальной колоды и запихиваем в перетасованную колоду  
 Random random = new Random();  
 int randomCardIndex = 0;  
 int originalSize = this.cards.size();  
 for(int i = 0; i<originalSize;i++){  
 //генерируем случайное число  
 randomCardIndex = random.nextInt((this.cards.size() - 1) + 1);  
 //засовываем новую карту во временную колоду  
 tmpDeck.add(this.cards.get(randomCardIndex));  
 //удаляем выбранную карту из старой колоды  
 this.cards.remove(randomCardIndex);  
 }  
 //перетасованную колоду помещаем в переменную с колодой  
 this.cards = tmpDeck;  
 }  
  
  
 //Убрать карту с колоды  
 public void removeCard(int i){  
 this.cards.remove(i);  
 }  
 //Взять карту с колоды  
 public Card getCard(int i){  
 return this.cards.get(i);  
 }  
  
 //Добавить карту в колоду  
 public void addCard(Card addCard){  
 this.cards.add(addCard);  
 }  
  
 //Взять верхнюю карту с колоды  
 public void draw(Deck comingFrom){  
 this.cards.add(comingFrom.getCard(0));  
 comingFrom.removeCard(0);  
 }  
  
 //Вывод колоды / руки  
 public String toString(){  
 StringBuilder cardListOutput = new StringBuilder();  
 int i = 0;  
 for(Card aCard : this.cards){  
 cardListOutput.append(" ").append(aCard.toString());  
 i++;  
 }  
 return cardListOutput.toString();  
 }  
  
 // Возвращаем все карты в колоду  
 public void moveAllToDeck(Deck moveTo){  
 int thisDeckSize = this.cards.size();  
 for(int i = 0; i < thisDeckSize; i++){  
 moveTo.addCard(this.getCard(i));  
 }  
 for(int i = 0; i < thisDeckSize; i++){  
 this.removeCard(0);  
 }  
 }  
  
 public int deckSize(){  
 return this.cards.size();  
 }  
  
 //Подсчет значения карт  
 public int cardsValue(){  
 int totalValue = 0;  
 int aces = 0;  
 //Для каждой карты в руке  
 for(Card aCard : this.cards){  
 //Switch по возможным значениям  
 switch (aCard.getValue()) {  
 case *TWO*, *JACK* -> totalValue += 2;  
 case *THREE*, *QUEEN* -> totalValue += 3;  
 case *FOUR*, *KING* -> totalValue += 4;  
 case *FIVE* -> totalValue += 5;  
 case *SIX* -> totalValue += 6;  
 case *SEVEN* -> totalValue += 7;  
 case *EIGHT* -> totalValue += 8;  
 case *NINE* -> totalValue += 9;  
 case *TEN* -> totalValue += 10;  
 case *ACE* -> aces += 1;  
 }  
 }  
  
 //Определяем общее значения карт с тузами  
 //Тузы или 11, или 1 - если туз = 11 будет превышать 21, то туз будет равен 1  
 for(int i = 0; i < aces; i++){  
 if (totalValue > 10){  
 totalValue += 1;  
 }  
 else{  
 totalValue += 11;  
 }  
 }  
  
 //Возврат значения  
 return totalValue;  
  
 }  
  
  
}

Card.java – карта

public class Card {  
  
 private final Suit suit;  
 private final Value value;  
  
 public Card(Suit suit, Value value){  
 this.suit = suit;  
 this.value = value;  
 }  
  
 public String toString(){  
 return this.suit.toString() + "-" + this.value.toString();  
 }  
  
 public Value getValue(){  
 return this.value;  
 }  
}

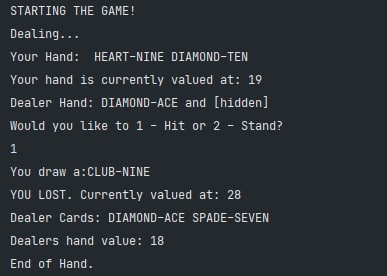
Suit.java – масть

public enum Suit {  
 *HEART*, *SPADE*, *DIAMOND*, *CLUB*}

Value.java – значення

public enum Value {  
 *TWO*, *THREE*, *FOUR*, *FIVE*, *SIX*, *SEVEN*, *EIGHT*, *NINE*, *TEN*, *JACK*, *QUEEN*, *KING*, *ACE*}

Результати роботи програми



Опис роботи програми

Запускаємо дві програми, спочатку програму серверу – Dealer.java, потім програму клієнта – Client.java, саме у такій послідовності. Після запуску клієнт під’єднується до сервера та отримує дані через InputStream, що передає сервер, а саме процес роздачі, отримані карти, значення карт та карти дилера. Далі очікується вибір користувача, а саме 2 команди – “Hit” та “Stand”, які зчитуються з консолі через InputStream та передаються до сервера через OutputStream. Далі сервер підраховує значення карт гравця та дилера, та виносить вердикт щодо фіналу гри.

Висновки

У результаті виконання лабораторної роботи було створено 2 програми (сервер та клієнт), які спілкуються між собою за допомогою сокетів та передають один між іншим дані для гри в «21» або BlackJack.

Ознайомилася сокетами та з особливостями передачі через InputStream/OutputStream, а саме проблемою, що виникла в мене – не передавалися дані. Згодом я дізналася, що після кожного рядка, що потрібно вивести, необхідно писати \n. Це було основною проблемою під час виконання лабораторної роботи.