# Задание 1.

package trainingTask1;

public class QuadricEquationsImpl{

public static void main(String[] args){

QuadricEquations.solveEquation(2,5,1);

QuadricEquations.solveEquation(1,2,1);

QuadricEquations.solveEquation(1,4,7);

}

}

class QuadricEquations{

//вложенный класс для рассчета дискриминанта

private static class Discriminant{

static double calculate(double a,double b,double c){

return b\*b-4\*a\*c;

}

}

//статический метод для решения квадратных уравнений с действительными коэффициентами

public static void solveEquation(double a,double b,double c){

double d = Discriminant.calculate(a,b,c);

if (d > 0) {

System.out.println("Уравнение имеет 2 действительных корня: х1 = "+((-b+Math.sqrt(d))/2\*a)+"; x2 = "+((-b-Math.sqrt(d))/2\*a));

}else if (d == 0){

System.out.println("Уравнение имеет 1 действительный корень: х = "+(-b/2\*a));

}else {

d \*= -1;

String b2 = ""+(-b/(2\*a));

String d2 = ""+Math.sqrt(d)/(2\*a)+"i";

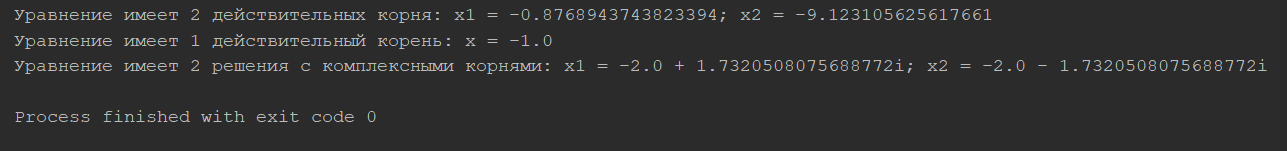
System.out.println("Уравнение имеет 2 решения с комплексными корнями: х1 = "+b2+" + "+d2+"; x2 = "+b2+" - "+d2);

}

}

}

Результат выполнения



# Задание 2.

package trainingTask1;

public class DiceGameImpl{

public static void main(String[] args){

int rnd = 1;

int N = 4;

int K = 3;

DiceGame game = new DiceGame();

game.initNewGame(N,K);

while (game.isActive()){

System.out.println("Round "+rnd);

game.nextRound();

rnd++;

System.out.println();

}

game.checkTable();

}

}

class DiceGame {

private int dicesNumber;

private Player[] players;

private boolean activeness = false;

//метод для инициализации начала игры

public void initNewGame(int N,int K){

this.dicesNumber = K;

players = new Player[N];

players[0] = new Player("You");

for (int i=1; i < N-1; i++){

players[i] = new Player("Player "+i);

}

players[N-1] = new Player("Comp");

activeness = true;

}

public boolean isActive(){

return this.activeness;

}

//метод поиска победителей

private static int findWinner(int[] cnt){

int index = 0;

int max = 0;

for (int i=0; i<cnt.length; i++){

if (max < cnt[i]){

max = cnt[i];

index = i;

}

}

return index;

}

//метод проверка единственности победителя

private static boolean isOneWinner(int[] cnt){

int max = 0;

int counts = 0;

for (int i=0; i<cnt.length; i++){

if (max < cnt[i]){max = cnt[i];}

}

for (int i=0; i<cnt.length; i++){

if (max == cnt[i]){counts++;}

}

if (counts==1) {return true;}

else {return false;}

}

//метод - проведение раунда

public void nextRound(){

if (!this.isActive()){

System.out.println("Игра не активна. Начните новую.");

return;

}

int[] cnt= new int[players.length];

int winner;

System.out.println("Суммы очков игроков:");

for (int j=0; j < players.length;j++){

for (int i=0; i < dicesNumber; i++){

cnt[j] += players[j].rollDices();

}

System.out.print(players[j].getName()+": "+cnt[j]+"; ");

}

System.out.println();

winner = DiceGame.findWinner(cnt);

if (!DiceGame.isOneWinner(cnt)) {

System.out.println("В этом раунде несколько игроков набрали одинаковое количество очков.");

}else {

System.out.println("В этом раунде победил игрок: " + players[winner].getName());

players[winner].addWin();

Player tmp = players[winner];

players[winner] = players[0];

players[0] = tmp;

if (players[0].getWins() == 7){

System.out.println("Игрок '" + players[winner].getName()+"' набрал 7 побед. На этом игра закончилась.");

activeness = false;

}

}

}

public void checkTable(){

System.out.println("Текущая ситуация по игре");

System.out.println("Игрок|Количество побед");

for (int i = 0; i<players.length; i++){

System.out.println(players[i].getName()+" | "+players[i].getWins());

}

}

}

class Player{

private int wins;

private String name;

public Player(){}

public Player(String name){

this.name = name;

}

public int getWins(){

return wins;

}

public void addWin(){

this.wins++;

}

public String getName(){

return this.name;

}

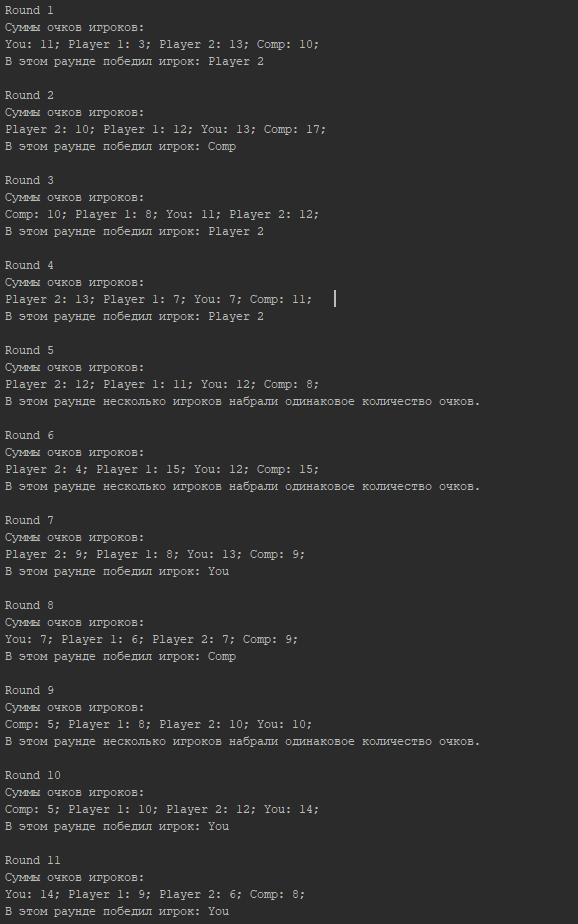
//имитация бросания кости

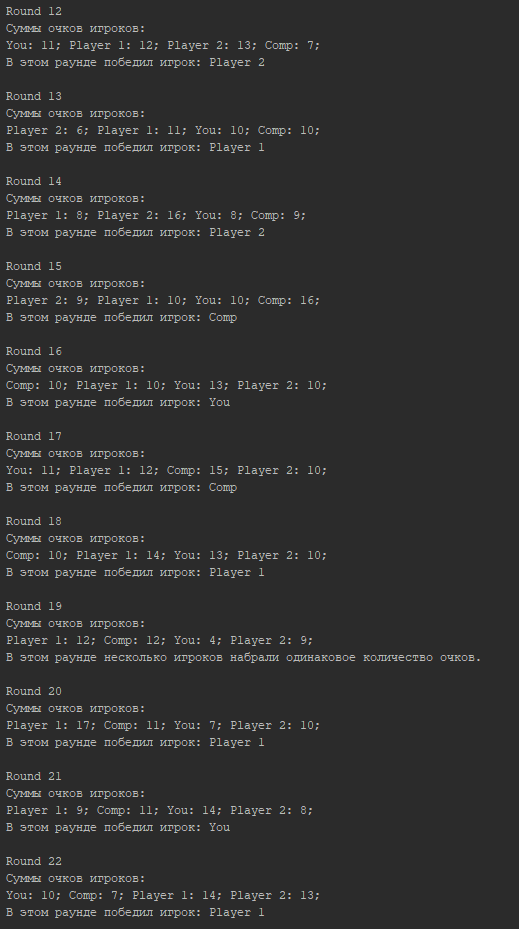
public int rollDices(){

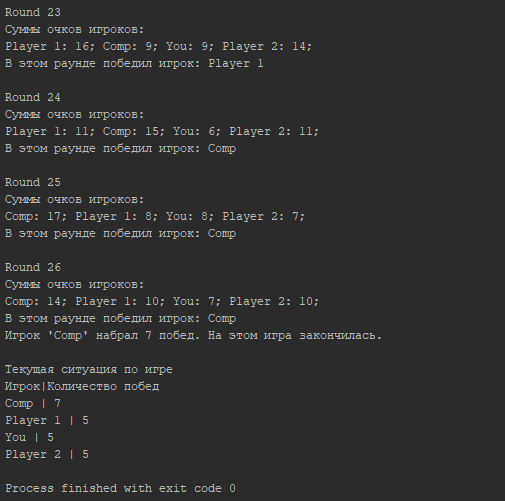
return (int) (Math.random() \* 6) + 1;

}

}







# Задание 3.

package trainingTask1;

import java.util.Calendar;

import java.util.GregorianCalendar;

import java.util.HashSet;

public class PersonAddressImpl {

//поиск позиции человека в массиве по по фамилии (первое вхождение) (если нет, то вернет null)

public static int searchSurname(Person[] persons, String surname){

int man = -1;

for (int i = 0; i<persons.length; i++){

if (persons[i].getSurname().equals(surname)) {

man = i;

break;

}

}

return man;

}

//поиск позиции человека в массиве по фамилии (вхождение под номером 'pos') (если нет, то вернет -1)

public static int searchSurname(Person[] persons, String surname, int pos){

int man = -1;

int cnt = 0;

for (int i = 0; i<persons.length; i++){

if (persons[i].getSurname().equals(surname)) {

cnt++;

if (cnt == pos) {

man = i;

break;

}

}

}

return man;

}

//поиск позиции человека в массиве по адресу(первое вхождение) (если такого нет, то вернет -1)

public static int searchAddress(Person[] persons, String address){

int man = -1;

for (int i = 0; i<persons.length; i++){

if (persons[i].getAddress().equals(address)) {

man = i;

break;

}

}

return man;

}

//поиск позиции человека в массиве по адресу (вхождение под номером 'pos') (если нет, то вернет -1)

public static int searchAddress(Person[] persons, String address, int pos){

int man = -1;

int cnt = 0;

for (int i = 0; i<persons.length; i++){

if (persons[i].getAddress().equals(address)) {

cnt++;

if (cnt == pos) {

man = i;

break;

}

}

}

return man;

}

//вывод людей, родившихся между определенными датами, (если таких нет, то null)

public static void searchDates(Person[] persons, Calendar d1, Calendar d2){

for (int i = 0; i<persons.length; i++){

if (persons[i].getBirthdate().compareTo(d1)>0 && persons[i].getBirthdate().compareTo(d2)<0){

System.out.println(persons[i].toString());

}

}

}

//поиск позиции самого молодого человека в массиве (первое вхождение)

public static int findYoung(Person[] persons){

Calendar young = persons[0].getBirthdate();

int index = 0;

for (int i = 0; i<persons.length; i++){

if (persons[i].getBirthdate().compareTo(young)>0) {

index = i;

young = persons[i].getBirthdate();

break;

}

}

return index;

}

//поиск позиции самого старого человека в массиве (первое вхождение)

public static int findOld(Person[] persons){

Calendar old = persons[0].getBirthdate();

int index = 0;

for (int i = 0; i<persons.length; i++){

if (persons[i].getBirthdate().compareTo(old)<0) {

index = i;

old = persons[i].getBirthdate();

break;

}

}

return index;

}

//вывод людей, проживающих на одной улице,и их индексов в массиве

public static void sameAddress(Person[] persons){

HashSet<String> strSet = new HashSet<>();

for (int i = 0; i<persons.length; i++){

strSet.add(persons[i].getAddress());

}

for (String addr : strSet) {

System.out.print(addr+": ");

for (int i = 0; i<persons.length; i++){

if (persons[i].getAddress().equals(addr)) {

System.out.print(persons[i].getFullName()+"; ");

}

}

System.out.println();

}

}

public static void main(String[] args){

Person[] persons = new Person[4];

persons[0] = new Person("Иван","Иванов");

persons[0].setBirthdate(11,12,1980);

persons[0].setAddress("Ул. Арбат");

persons[1] = new Person("Евгений","Петров");

persons[1].setBirthdate(11,12,1978);

persons[1].setAddress("Ул. Смоленская");

persons[2] = new Person("Сергей","Иванов");

persons[2].setBirthdate(1,12,1988);

persons[2].setAddress("Ул. Арбат");

persons[3] = new Person("Алексей","Семенченко");

persons[3].setBirthdate(10,7,1985);

persons[3].setAddress("Ул. Смоленская");

System.out.println("проверка подпункта 1");

System.out.println(PersonAddressImpl.searchSurname(persons,"Петров"));

System.out.println(PersonAddressImpl.searchSurname(persons,"Иванов",2)+"\n");

System.out.println("проверка подпункта 2");

System.out.println(PersonAddressImpl.searchAddress(persons,"Ул. Смоленская"));

System.out.println(PersonAddressImpl.searchAddress(persons,"Ул. Смоленская",2)+"\n");

System.out.println("проверка подпункта 3");

Calendar d1 = new GregorianCalendar(1983,1,1);

Calendar d2 = new GregorianCalendar(2000,1,1);

PersonAddressImpl.searchDates(persons,d1,d2);

System.out.println();

System.out.println("проверка подпункта 4");

System.out.println(PersonAddressImpl.findYoung(persons));

System.out.println(PersonAddressImpl.findOld(persons)+"\n");

System.out.println("проверка подпункта 5");

PersonAddressImpl.sameAddress(persons);

System.out.println();

}

}

//кдасс для сущности человека

class Person{

private String name;

private String surname;

private Calendar birthdate;

private Address address;

public Person(){}

public Person(String name){

this.name = name;

}

public Person(String name,String surname){

this.name = name;

this.surname = surname;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getSurname() {

return surname;

}

public void setSurname(String surname) {

this.surname = surname;

}

public String getFullName() {

return name+" "+surname;

}

public Calendar getBirthdate() {

return birthdate;

}

public void setBirthdate(int day,int month, int year) {

Calendar birthdate = new GregorianCalendar(year,month,day);

this.birthdate = birthdate;

}

public String getAddress() {

return address.getName();

}

public void setAddress(String address) {

this.address = new Address();

this.address.setName(address);

}

@Override

public String toString() {

return name + ", " + surname + ", " +

birthdate.get(Calendar.DATE)+"."+(birthdate.get(Calendar.MONTH)+1)+"."+birthdate.get(Calendar.YEAR)+

", " + address.getName()+ ";";

}

}

//класс для сущности адреса

class Address{

private String name;

public String getName() {

return name;

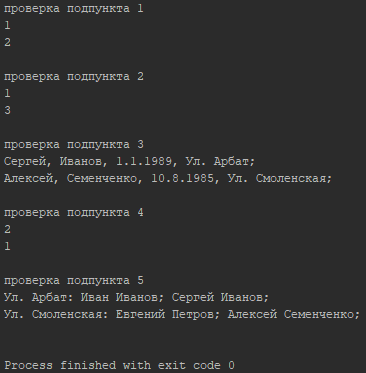
}

public void setName(String name) {

this.name = name;

}

}



# Ссылка на GitHub

[yagorkas/Training-Tasks (github.com)](https://github.com/yagorkas/Training-Tasks)