

Задача1. Должен быть реализован класс для решения квадратных уравнений. Вычисление дискриминанта должен выполнять вложенный класс.

### Код программы:

#### Класс Lab1

```
package lab1;
public class Lab1 {
    public static void main(String[] args) {
        //Создаем объект класса
        QuadraticEquation qe = new QuadraticEquation();
        QuadraticEquation.Discriminant discriminant =
            qe.new Discriminant();
        //Вызываем метод класса квадратичного уравнения
        qe.scan();
        discriminant.calc();
    }
}
```

#### Класс QuadraticEquation

```
package lab1;
import java.util.Scanner;

public class QuadraticEquation {
    double a, b, c;
    //Метод для сканирования данных ввода с клавиатуры
    public void scan() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Введите значение a: ");
        a = scanner.nextDouble();
        System.out.print("Введите значение b: ");
        b = scanner.nextDouble();
        System.out.print("Введите значение c: ");
        c = scanner.nextDouble();
    }

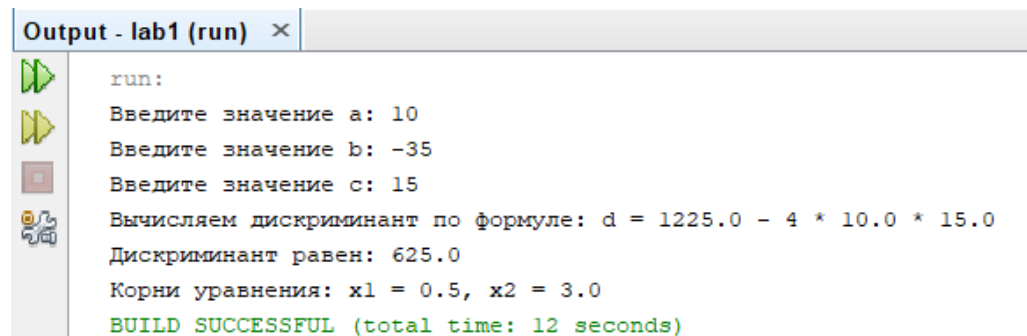
    public class Discriminant {
        //Метод для вычислений дискриминанта
        public void calc() {
            System.out.println("Вычисляем дискриминант по формуле:  $d = b^2 - 4ac$ ");
            double d = b * b - 4 * a * c;
            System.out.println("Дискриминант равен: " + d);
            if (d > 0) {
                double x1 = (-b - Math.sqrt(d)) / (2 * a);
                double x2 = (-b + Math.sqrt(d)) / (2 * a);
                System.out.println("Корни уравнения:  $x_1 =$ " + x1 + ",  $x_2 =$ " + x2);
            } else if (d == 0) {
                double x;
                x = -b / (2 * a);
            }
        }
    }
}
```

```

        System.out.println("Уравнение имеет единственный
корень: x = " + x);
    } else {
        System.out.println("Уравнение не имеет
действительных корней");
    }
}
}
}

```

Вывод:



```

run:
Введите значение a: 10
Введите значение b: -35
Введите значение c: 15
Вычисляем дискриминант по формуле: d = 1225.0 - 4 * 10.0 * 15.0
Дискриминант равен: 625.0
Корни уравнения: x1 = 0.5, x2 = 3.0
BUILD SUCCESSFUL (total time: 12 seconds)

```

Таким образом, Вложенный класс Discriminant был создан для написания вспомогательного кода для класса QuadraticEquation. Область видимости вложенного класса ограничена областью видимости внешнего класса. Вложенные классы позволяют группировать классы, логически принадлежащие друг другу, и управлять доступом к ним. Нестатические вложенные классы называют также внутренними классами. Внутренний класс Discriminant создан внутри окружающего класса и имеет доступ ко всем переменным и методам своего внешнего класса QuadraticEquation и может непосредственно ссылаться на них.

Задача2. Создание игры в кости. Участвуют N игроков (компьютер — последний в списке). Одновременно бросаются K кубиков. Выигрывает игрок с наибольшей суммой очков. Тот, кто победил, бросает кубики первым в следующем раунде. Игра продолжается до достижения 7 побед. Начало игры — за пользователем.

### Код программы на Java:

#### Главный метод main:

```
public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    int N = 0, K = 0, winnerInRound = 0, roundNum = 0,
numWin = 0, indWin = 0, indxComp=0;
    //ввод с клавиатуры количества игроков и кубиков
    System.out.print("Введите кол-во игроков: ");
    K = in.nextInt();
    System.out.print("Введите кол-во кубиков: ");
    N = in.nextInt();
    indxComp=N-1; //индекс компьютера
    ArrayList<Player> players = new
    ArrayList<Player>(N); //массив игроков
    for (int i = 0; i < N ; i++) {
        players.add(i, new Player(0, 0));
    }
    if (K > 1 && N > 1) { //игроков должно быть больше 1
        //игра продолжается до тех пор, пока кол-во
        //выигрышей одного из игроков не составит 7
        while (numWin != 7) { //до 7 выигрышей одного из
            игроков
                System.out.println("\nИгра № " + roundNum);
                                //номер раунда

                //очистить очки предыдущего раунда у игроков
                for (int i = 0; i < players.size(); i++) {
                    players.get(i).setsum(0); //сумма очков
                    каждого игрока = 0
                }

                //игрок, победивший в прошлом раунде кидает
                первым

                players.get(winnerInRound).setsum(rollTheDice(K));
                //вывод информации об игроке
                System.out.println("Первым делает бросок:
"+printInformPlayer(winnerInRound, indxComp));
                printPlayer(players, winnerInRound, indxComp);
                //вывод информации об игроке

                //игроки, не победившие в прошлом раунде, кидают
                по очереди
            }
        }
    }
```

```

        for (int i = 0; i < players.size(); i++) {
            //ход каждого игрока
            if (i == winnerInRound ) {
                continue; //игрок, победивший в
                           прошлом раунде уже сделал бросок
            }
            players.get(i).setsum(rollTheDice(K));
            //запись суммы очков в поле sumscore игрока
            printPlayer(players, i, indxComp); //вывод
                                                информации об игроке
        }

        //поиска игрока с самым большим кол-вом очков
        winnerInRound = playerMaxSum(players); //номер
                                                победителя в раунде
        //нашли игрока с наибольшим кол-вом очков
        players.get(winnerInRound).setwin(1); // win+=1
                                                - игрок победил в раунде

        //вывод победителя в раунде
        System.out.println("\nПобедил: " +
            printInformPlayer(winnerInRound, indxComp) + "
            его очки: " +
            players.get(winnerInRound).getsum() + " кол-во
            его побед: " +
            players.get(winnerInRound).getwin());
        //наибольшее кол-во побед у какого-либо игрока
        indWin = playerWinner(players); //индекс игрока
                                                с наибольшим кол-вом побед
        numWin = players.get(indWin).getwin(); //кол-во
                                                побед
        roundNum++;
    }
    //номер победителя в игре
    System.out.print("Победитель в игре: " +
        printInformPlayer(indWin, indxComp) + " кол-во его
        побед: " + players.get(indWin).getwin());
} else {
    System.out.println("Игроков должно быть больше 1");
}
}

```

## Метод поиска игрока с самым большим кол-вом очков

```

//метод поиска игрока с самым большим кол-вом очков
public static int playerMaxSum(ArrayList<Player> players) {
    int maxsum = players.get(0).getsum(), winnerInRound = 0,
    sum = 0;
    for (int j = 1; j < players.size(); j++) { // проход по
    массиву игроков

```

```

        sum = players.get(j).getsum();
        if (sum > maxsum) {
            winnerInRound = j; //номер игрока с наибольшим
КОЛ-ВОМ ОЧКОВ
            maxsum = sum;
        }
    }
    return winnerInRound;
}

```

### Метод поиска игрока с самым большим кол-вом побед

```

//метод поиска игрока с самым большим кол-вом побед
public static int playerWinner(ArrayList<Player> players) {
    int maxwin = players.get(0).getwin(), winnerIngame = 0,
win = 0;
    for (int j = 0; j < players.size(); j++) { // проход по
массиву игроков
        win = players.get(j).getwin();
        if (win > maxwin) {
            winnerIngame = j; //номер игрока с наибольшим
КОЛ-ВОМ побед
        }
    }
    return winnerIngame;
}

```

### Метод броска кубика К раз

```

//метод броска кубика К раз
public static int rollTheDice(int K) {
    int step = 0, sumscore = 0;

    while (step < K) { //бросать кубик К раз
        sumscore += (int) (Math.random() * 6 + 1); //сумма
очков всех бросков одного игрока
        step++; //кол-во бросков
    }
    return sumscore; //сумма очков всех бросков
}

```

### Метод вывода игрока и очков

```

//метод вывода игрока и его очков
public static void printPlayer(ArrayList<Player> players,
int indx, int indxComp) {
    if (indx == 0) {
        System.out.print("У вас очков: "+
players.get(indx).getsum() + " | ");
    }
    if (indx == indxComp) {

```

```

        System.out.print("У компьютера очков: " +
players.get(indx).getsum() + " | ");
    }
    if (indx != 0 && indx != indxComp){
        System.out.print("У игрока: "+indx+" очков: " +
players.get(indx).getsum() + " | ");
    }
}

```

## Метод вывода информации об игроке

```

//метод вывода информации об игроке
public static String printInformPlayer(int indx, int
indxComp) {
    String str = new String();
    if (indx == 0) {
        str="Пользователь";
    }
    if (indx == indxComp) {
        str="Компьютер";
    }
    if (indx != 0 && indx != indxComp){
        str="Игрок: "+indx;
    }
    return str;
}

```

## Метод класс игрока

```

static class Player {

    private int win; //кол-во побед игрока
    private int sumscore; //сумма очков за раунд

    Player(int win, int sumscore) {
        this.win = win;
        this.sumscore = sumscore;
    }
    public void setsum(int sumscore) {
        this.sumscore = sumscore;
    }
    public int getsum() {
        return sumscore;
    }
    public void setwin(int win) {
        this.win += win;
    }
    public int getwin() {
        return win;
    }
}

```

При запуске программы пользователю необходимо ввести количество игроков и кубиков.

Первым игроком считается пользователем, а последним – компьютер. В начале игры первым ходит пользователь, а затем тот, кто победил в предыдущем раунде, набрав большее кол-во очков.

Пользователю сообщается кол-во очков каждого игрока, а также информация о победителе в текущем раунде: номер игрока, его очки в этом раунде, общее кол-во его побед за всю игру.

```
run:
Введите кол-во игроков: 3
Введите кол-во кубиков: 3

Игра № 0
Первым делает бросок: Пользователь
У вас очков: 8 | У игрока: 1 очков: 11 | У компьютера очков: 9 |
Победил: Игрок: 1 его очки: 11 кол-во его побед: 1

Игра № 1
Первым делает бросок: Игрок: 1
У игрока: 1 очков: 6 | У вас очков: 10 | У компьютера очков: 14 |
Победил: Компьютер его очки: 14 кол-во его побед: 1

Игра № 2
Первым делает бросок: Компьютер
У компьютера очков: 14 | У вас очков: 17 | У игрока: 1 очков: 10 |
Победил: Пользователь его очки: 17 кол-во его побед: 1

Игра № 3
Первым делает бросок: Пользователь
У вас очков: 13 | У игрока: 1 очков: 8 | У компьютера очков: 11 |
Победил: Пользователь его очки: 13 кол-во его побед: 2
```

Игра продолжается до тех пор, пока один из игроков не одержит победу 7 раз, тогда игра завершается, и выводится информация о победителе.

```
Игра № 13
Первым делает бросок: Игрок: 1
У игрока: 1 очков: 6 | У вас очков: 7 | У компьютера очков: 13 |
Победил: Компьютер его очки: 13 кол-во его побед: 2

Игра № 14
Первым делает бросок: Компьютер
У компьютера очков: 5 | У вас очков: 12 | У игрока: 1 очков: 13 |
Победил: Игрок: 1 его очки: 13 кол-во его побед: 7
Победитель в игре: Игрок: 1 кол-во его побед: 7
```

Задача3. Напишите программу «Адрес человека». Создайте сущность "Человек", которая связана с сущностью "Адрес". Предполагается, что у каждого человека может быть только один адрес. Создайте массив объектов "Человек" (не менее четырех) и выполните следующие операции по массиву:

- Найдите человека по фамилии.
- Найдите человека по адресу или его атрибуту.
- Выведите людей, родившихся в определенный период.
- Определите самого старого и самого молодого человека.
- Найдите людей, проживающих на одной улице.

Код программы на Java:

Главный метод main:

```
public static void main(String[] args) {
    formater = new SimpleDateFormat("dd.MM.yyyy"); //представление
    даты в удобном формате
    sc = new Scanner(System.in, "windows-1251"); //сканер для
    консольного ввода

    ArrayList<Address> addressList = new ArrayList<Address>(3);
    //массив адрессов
    ArrayList<Humans> humansList = new ArrayList<Humans>(3);
    //массив жителей

    //заполнение массива адрессов
    addressList.add(new Address("Самара", "Полевая", 1));
    addressList.add(new Address("Самара", "Дачная", 2));
    addressList.add(new Address("Самара", "Гагарина", 3));
    addressList.add(new Address("Самара", "Полевая", 4));
    addressList.add(new Address("Самара", "Дачная", 5));

    //заполнение массива людей
    humansList.add(new Humans("Николай", "Петров", 0, new
    GregorianCalendar(1983, 3, 23)));
    humansList.add(new Humans("Людмила", "Зайцева", 1, new
    GregorianCalendar(1986, 6, 26)));
    humansList.add(new Humans("Татьяна", "Николаева", 2, new
    GregorianCalendar(1989, 9, 29)));
    humansList.add(new Humans("Петр", "Крышев", 3, new
    GregorianCalendar(1997, 7, 27)));
    humansList.add(new Humans("Валентина", "Кошкина", 4, new
    GregorianCalendar(1987, 7, 7)));

    //вывод возможных действий
    System.out.println(" Выберите действие из списка: \n"
        + "1 - Вывод всех жителей;\n"
        + "2 - Поиск человека по фамилии;\n"
        + "3 - Поиск человека по адресу;\n"
        + "4 - Вывод людей, родившихся между определенными
        датами;\n"
```



```

+ "5 - Вывод самого молодого жителя;\n"
+ "6 - Вывод людей, проживающих на одной улице;\n"
+ "--введите любое целое число для завершения;");

//выбор действия в консоли
boolean goOn = true; //продолжать предлагать действия
while (goOn) {
    System.out.print("\nВведите номер выбранного действия: ");
    try {
        int numAction = sc.nextInt(); // ввод с консоли номера
        действия
        sc.nextLine();
        switch (numAction) {
            case 1: //Вывод всех жителей
                printHumansAll(humansList, addressList);
                break;
            case 2: //Поиск человека по фамилии
                System.out.print("Введите фамилию: ");
                String surname = sc.nextLine(); //ввод фамилии
                с консоли
                searchByLastname(humansList, addressList,
                    surname);
                break;
            case 3: //Поиск человека по адресу
                System.out.print("Введите индекс адреса: ");
                try { //перехват ошибки ввода не целого числа
                    int indxAddress = sc.nextInt(); //ввод с
                    консоли
                    printHumanAtAddress(humansList,
                        addressList, indxAddress);
                } catch (InputMismatchException e) {
                    System.out.println("Вы ввели не число");
                }
                sc.nextLine();
                break;
            case 4: //Вывод людей, родившихся между
                определенными датами
                GregorianCalendar startDate, stopDate;
                //начальная и конечная даты
                try { //проверка на ввод целых чисел
                    System.out.println("Введите начальную дату
                    в формате 01.01.1900: ");
                    startDate = inputDate();
                    System.out.println("Введите конечную дату
                    в формате 01.01.1900: ");
                    stopDate = inputDate();
                    printBetweenDates(humansList, addressList,
                        startDate, stopDate);
                } catch (InputMismatchException e) {
                    System.out.println("Вы ввели не целое
                    число");
                    sc.nextLine();
                    break;
                }
                break;
            case 5: //Вывод самого молодого жителя
                printOldest(humansList, addressList);

```

```

        break;
    case 6: //Вывод людей, проживающих на одной улице
        System.out.print("Введите название улицы: ");
        String streetName = sc.nextLine(); //ввод
            названия улицы
        printLiveOnStreet(humansList, addressList,
            streetName);
        break;
    default: //введено любое целое число не из списка
        действий
        System.out.println("Пока-пока.");
        goOn = false;
        break;
    }
} catch (InputMismatchException e) {
    System.out.println("Вы ввели не целое число");
    sc.nextLine();
}
}
}

```

Метод вывода данных о человеке и его адресе:

```

//вывод одного человека
static void printHuman(Humans human, ArrayList<Address>
    addressList) {
    System.out.println("По адресу:" + " ул." +
        addressList.get(human.address).streetName
            + ", д." +
        addressList.get(human.address).houseNumber + " проживает:
        "
            + human.surname + " " + human.name + ", " +
        formater.format(human.dateOfBirth.getTime()));
}

```

Метод вывода списка всех жителей:

```

//вывод всех жителей
public static void printHumansAll(ArrayList<Humans>
    humansList, ArrayList<Address> addressList) {
    for (Humans human : humansList) {
        printHuman(human, addressList);
    }
}

```

Метод поиска человека по фамилии:

```

//поиск человека по фамилии
static void searchByLastname(ArrayList<Humans> humansList,
    ArrayList<Address> addressList, String surname) {
    boolean found = false; //по-умолчанию человек не найден
    for (Humans human : humansList) {

```

```

        if
            (human.surname.toLowerCase().contains(surname.toLower
            rCase())) {
                printHuman(human, addressList); //вывод человека
                found = true; //человек найден
                break;
            }
        }
        if (!found) {
            System.out.println("Человек не найден");
        }
    }
}

```

Метод поиска человека по адресу:

```

//вывод человека по адресу
public static void printHumanAtAddress(ArrayList<Humans>
humansList, ArrayList<Address> addressList, int indxAddress) {
    boolean found = false; //по-умолчанию человек не найден
    for (Humans human : humansList) {
        if (human.address == indxAddress) { //индекс адреса
            человека совпадает с заданным
            printHuman(human, addressList); //вывод человека
            found = true; //человек найден
            break;
        }
    }
    if (!found) {
        System.out.println("Адрес не найден");
    }
}

```

Метод вывода людей, родившихся между определенными датами:

```

//вывод людей, родившихся между определенными датами
public static void printBetweenDates(ArrayList<Humans>
humansList, ArrayList<Address> addressList, GregorianCalendar
minDate, GregorianCalendar maxDate) {
    for (Humans human : humansList) {
        if (human.dateOfBirth.after(minDate) &&
            human.dateOfBirth.before(maxDate) ||
            human.dateOfBirth.before(minDate) &&
            human.dateOfBirth.after(maxDate)) {
            printHuman(human, addressList); //вывод человека
        }
    }
}

```

Метод вывода самого молодого человека:

```

//вывод самого молодого человека
public static void printOldest(ArrayList<Humans> humansList,
ArrayList<Address> addressList) {

```

```

        Calendar maxDate = humansList.get(0).dateOfBirth;
        //первый житель принимается за самого молодого
        int count = 0, i = 0;
        for (Humans human : humansList) { //проход по массиву
            всех жителей
            if (human.dateOfBirth.after(maxDate)) { //сравнение
                дат рождений жителей
                maxDate = human.dateOfBirth;
                count = i;
            }i++;
        }
        System.out.print("Самый молодой житель: ");
        printHuman(humansList.get(count), addressList); //вывод
        самого молодого человека
    }
}

```

Метод вывода людей, проживающих на одной улице:

```

//вывод людей, проживающих на одной улице
public static void printLiveOnStreet(ArrayList<Humans>
humansList, ArrayList<Address> addressList, String streetName) {
    int i = 0;
    boolean found = false; //люди не найдены
    for (Address adr : addressList) { //проход по массиву
        адресов
        if
            (adr.streetName.toLowerCase().contains(streetName.toL
owerCase())) { //найжены адреса с заданной улицей
            for (int j = 0; j < humansList.size(); j++) {
                //проход по массиву людей
                if (humansList.get(j).address == i) {
                    //найден человек с индексом адреса заданной
                    улицы
                    printHuman(humansList.get(j),
                        addressList); //вывод человека
                    found = true; //найден человек
                    break;
                }
            }
        } i++
    }
    if (!found) {
        System.out.println("Жители не найдены");
    }
}

```

Вспомогательный метод ввода даты:

```

//консольный ввод даты
static GregorianCalendar inputDate() {
    int d = 0, m = 0, y = 0;
    //try{
    System.out.print("день: ");
}

```

```

        d = sc.nextInt();
        System.out.print("месяц: ");
        m = sc.nextInt();
        System.out.print("Год: ");
        y = sc.nextInt();
        return new GregorianCalendar(y, m, d);
    }

```

Классы адреса и человека:

```

//адрес
public static class Address {
    String streetName; //название улицы
    int houseNumber; //номер дома
    private Address(String city, String streetName, int
houseNumber) {
        this.streetName = streetName;
        this.houseNumber = houseNumber;
    }
}
//человек
public static class Humans {
    String name; //имя
    String surname; //фамилия
    int address; //индекс адреса
    Calendar dateOfBirth; //дата рождения
    private Humans(String name, String surname, int address,
Calendar dateOfBirth) {
        this.name = name;
        this.surname = surname;
        this.address = address;
        this.dateOfBirth = dateOfBirth;
    }
}

```

Результат работы программы:

Программа осуществляет работу с пользователем посредством консольного ввода-вывода. При запуске программы пользователю предлагается выбрать действие из выведенного на экран списка действий.

При выборе действия 1 на экран выводится список всех жителей.

run:

Выберите действие из списка:

- 1 - Вывод всех жителей;
- 2 - Поиск человека по фамилии;
- 3 - Поиск человека по адресу;
- 4 - Вывод людей, родившихся между определенными датами;
- 5 - Вывод самого старого жителя;
- 6 - Вывод людей, проживающих на одной улице;
- введите любое целое число для завершения;

Введите номер выбранного действия: 1

По адресу: ул.Полевая, д.1 проживает: Петров Николай, 23.04.1983

По адресу: ул.Дачная, д.2 проживает: Зайцева Людмила, 26.07.1986

По адресу: ул.Гагарина, д.3 проживает: Николаева Татьяна, 29.10.1989

По адресу: ул.Полевая, д.4 проживает: Крышев Петр, 27.08.1997

По адресу: ул.Дачная, д.5 проживает: Кошкина Валентина, 07.08.1987

При выборе действия 2 пользователь вводит фамилию человека, по которой осуществляется поиск жителя с заданной фамилией.

При выборе действия 3 происходит поиск жителя по индексу его адреса.

Введите номер выбранного действия: 2

Введите фамилию: Зайцева

По адресу: ул.Дачная, д.2 проживает: Зайцева Людмила, 26.07.1986

Введите номер выбранного действия: 3

Введите индекс адреса: 1

По адресу: ул.Дачная, д.2 проживает: Зайцева Людмила, 26.07.1986

При выборе действия 4 пользователю необходимо ввести две даты. После ввода начальной и конечной даты, осуществляется поиск и вывод всех жителей, родившихся между заданными датами.

Введите номер выбранного действия: 4

Введите начальную дату в формате 01.01.1900:

день: 1

месяц: 1

Год: 1980

Введите конечную дату в формате 01.01.1900:

день: 1

месяц: 1

Год: 1990

По адресу: ул.Полевая, д.1 проживает: Петров Николай, 23.04.1983

По адресу: ул.Дачная, д.2 проживает: Зайцева Людмила, 26.07.1986

По адресу: ул.Гагарина, д.3 проживает: Николаева Татьяна, 29.10.1989

По адресу: ул.Дачная, д.5 проживает: Кошкина Валентина, 07.08.1987

При выборе действия 5 происходит поиск и вывод самого молодого жителя.

При выборе действия 6 пользователь вводит название улицы и получает список людей, проживающей на ней.

Программа просит выбрать действие до тех пор, пока не будет введено любое целое число, не относящееся к списку действий, например 10.

```
Введите номер выбранного действия: 5
Самый молодой житель: По адресу: ул.Полевая, д.4 проживает: Крышев Петр, 27.08.1997

Введите номер выбранного действия: 6
Введите название улицы: Полевая
По адресу: ул.Полевая, д.1 проживает: Петров Николай, 23.04.1983
По адресу: ул.Полевая, д.4 проживает: Крышев Петр, 27.08.1997

Введите номер выбранного действия: 10
Пока-пока.
```

В случае ввода неверных чисел, пользователь получает соответствующие сообщения.

```
Введите номер выбранного действия: в
Вы ввели не целое число
```