1. Цель работы.

В ходе выполнения лабораторной работы нам нужно выполнить программную реализацию экспертной системы/системы принятия решения.

2. Ход работы.

2.1. Формирование базы вопросов.

Чтобы сделать выборку наиболее эффективной, будем задавать пользователю четыре следующих, ключевых вопроса:

1) Есть ли у тебя аллергия?
2) Есть ли у тебя опыт в заведении домашних питомцев?
3) Сколько времени в день ты готов уделять питомцу?
4) Сколько денег планируется потратить?

Таким образом, мы получим достаточное количество информации, чтобы дальше наша экспертная система выдала результат. Если же не задавать эти вопросы, то получится, что решение формируется для всех пользователей одинаково, исходя из просто общего рейтинга, что в корни неправильно - система должна быть гибкой, всё должно быть индивидуально.

В дальнейшем, при разработке интерфейса планируется сделать окно с этими вопросами, и кнопкой запроса решения. При нажатии на неё сервер будет обрабатывать наши данные, и выдавать соответственно 3 рекомендуемых животных и 3 набора товаров, которые являются наиболее покупаемыми. Также планируется обработка некорректного ввода данных.

2.2. Организация базы данных.

В целях оптимизации разработки программы в качестве БД будет выступать текстовый документ формата txt. Итак, о наборах мы будем хранить следующую информацию: название, стоимость, количество проданных, средняя оценка пользователя. То есть, каждый раз, когда пользователь будет покупать новый набор, мы будем просить его поставить оценку, и обновлять данные в БД: количество продаж будет увеличено на 1, а средняя оценка будет обновлена в соответствии с последней.

Если конкретизировать, то мы будем использовать формулу $\frac{N \cdot R + L}{(N+1)}$, где N–прошлое количество продаж, R–прошлая средняя оценка, L–последняя оценка. Эту же формулу мы будем использовать для вычисления средней оценки каждого из питомцев.

Итак, о питомцах мы будем хранить следующую информацию: название питомца, описание, возможно ли содержать людям с аллергией, возможно ли содержать без опыта в заведении питомцев, количество часов, требуемых на содержание питомца (в день), стоимость, количество продаж и средняя оценка пользователей. Выглядит это всё следующим образом:

Рис. 1. Хранение информации о наборах.

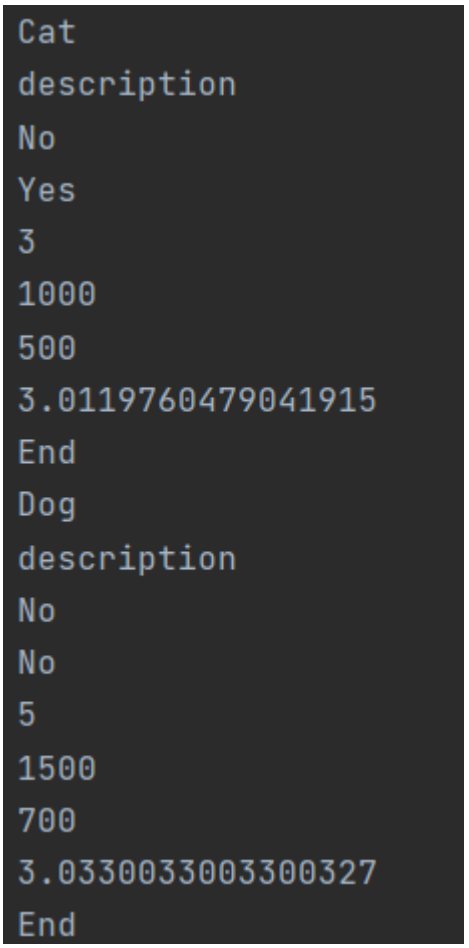

Рис. 2. Хранение информации о животных.

2.3. Реализация алгоритма принятия решения.

Как уже говорилось в пункте 2.2, нам нужно учитывать предпочтения пользователя помимо общего рейтинга того или иного питомца, поэтому будем пересчитывать рейтинг каждого из них под определённого пользователя. Наиболее значимым вопросом является первый: “Есть ли у тебя какая-то аллергия?”. Если пользователь отвечает на нём “Да”, то система минимизирует рейтинг тех животных, которые не подходят под данное условие, то есть их нельзя содержать людям с аллергией. Эту информацию система получает из базы данных и проверяет, проходясь по каждому из питомцев. Например: как бы ни была популярна кошка по продажам и по среднему рейтингу, если пользователь ответит, что у него есть аллергия – мы никогда ему её не предложим.

По другим всё не так “строго”. Вторым по значимости вопросом является вопрос о наличии опыта, и третьим - количество свободного времени. Это можно объяснить тем, что если первый вопрос отвечает за безопасность хозяина, то последние два отвечают за безопасность самого

питомца. Вопрос цены же не является таким критичным по сравнению с первыми двумя. Например, если животному требуется хозяин с опытом в заведении питомцев (это система узнаёт из базы данных), то мы “сокращаем” ему рейтинг примерно в два раза. Что касается вопросов свободного времени и стоимости, то мы вычитаем из рейтинга переменную, напрямую связанную с разницей между “желаемой” и “требуемой” характеристиками питомца.

2.4. Подключение интерфейса к программе.

Итак, остаётся только оформить разработанную нами систему. На главном экране должны быть вопросы, кнопка запроса решения. Далее должны быть результаты работы. В итоге имеем:

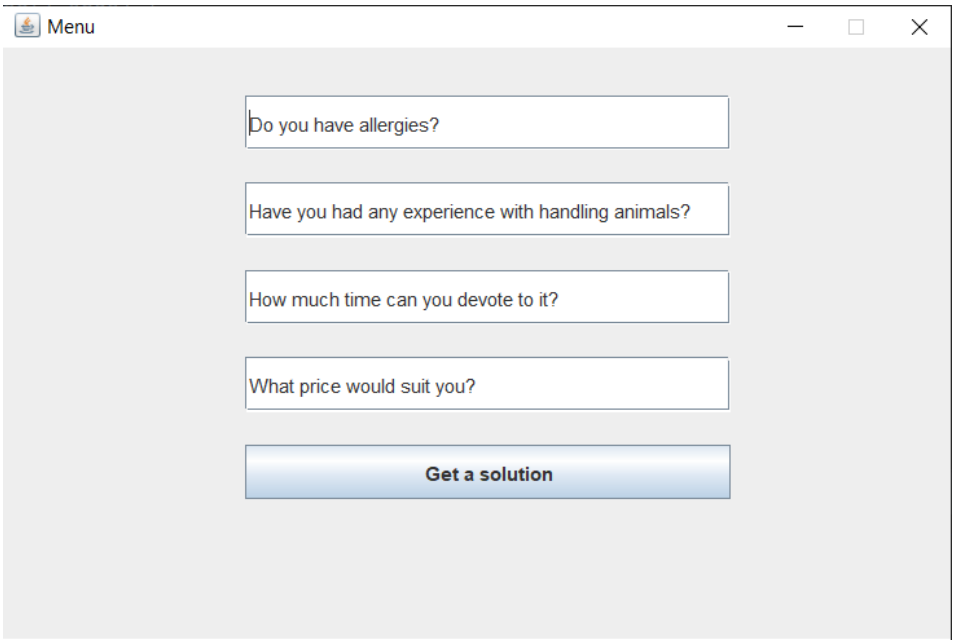

Рис. 3. Меню.

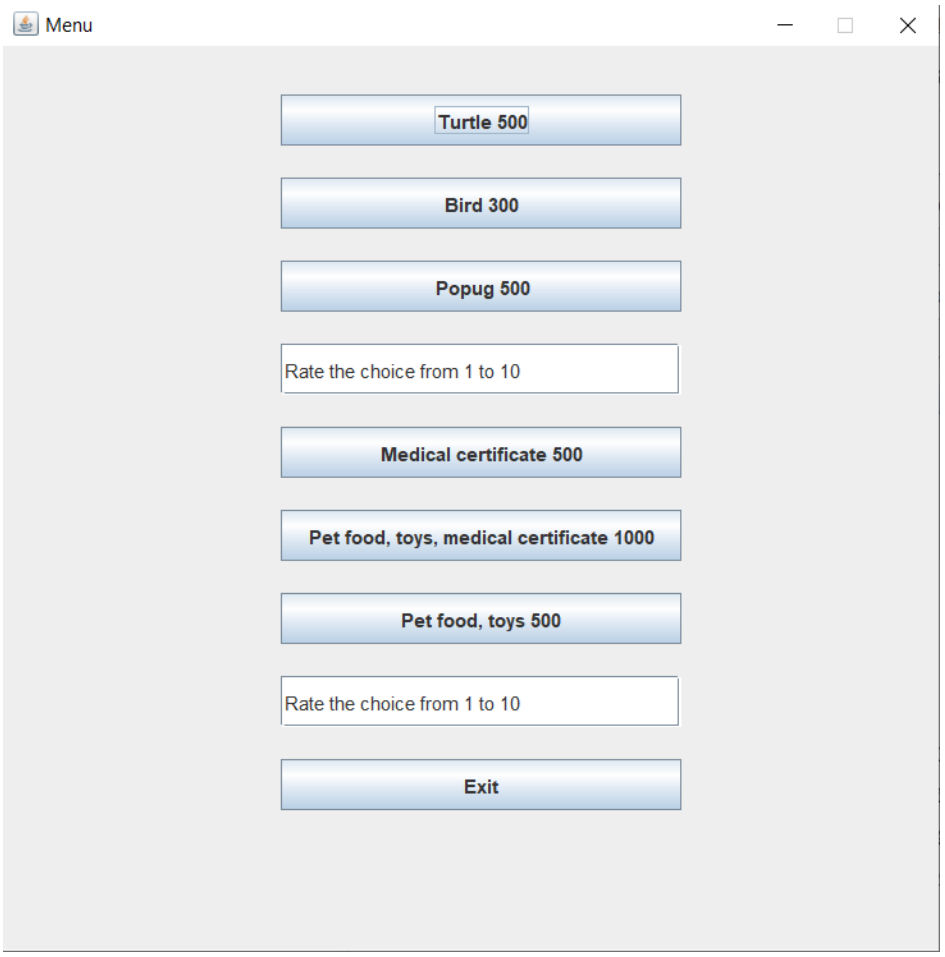

Рис. 4. Результат работы системы.

Рядом с названием питомцев и наборов написана их стоимость.

2.5. Тестирование работы системы.

Итак, протестируем работу программы. Для начала попробуем сразу отправить решение, ничего не поменяв:

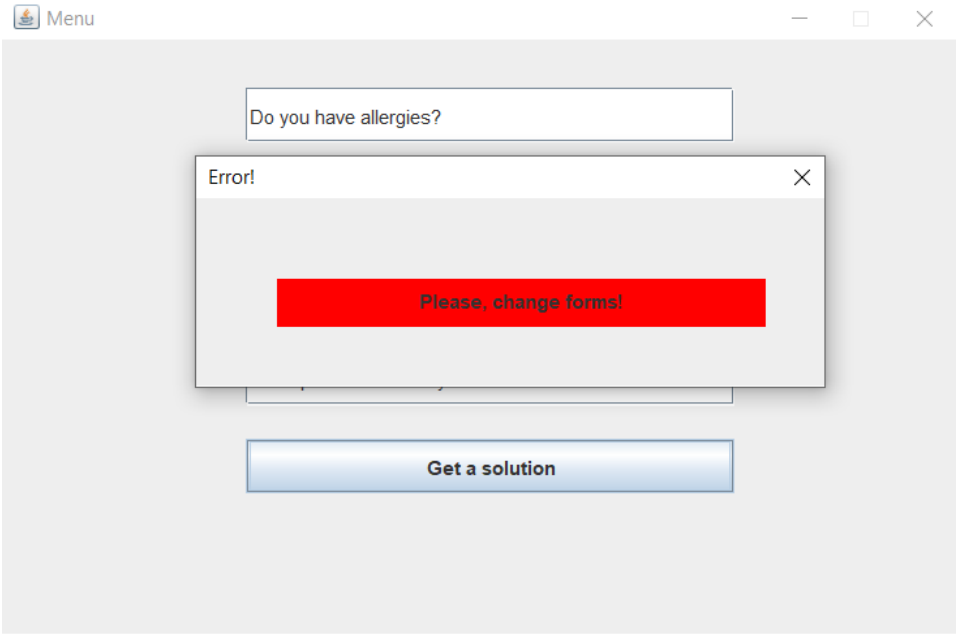

Рис. 5. Обработка ошибки в входных данных.

Как мы видим, программа отреагировала корректно. Пока не закроем уведомление, ничего менять нельзя. Вбиваем корректные данные и производим отправку:

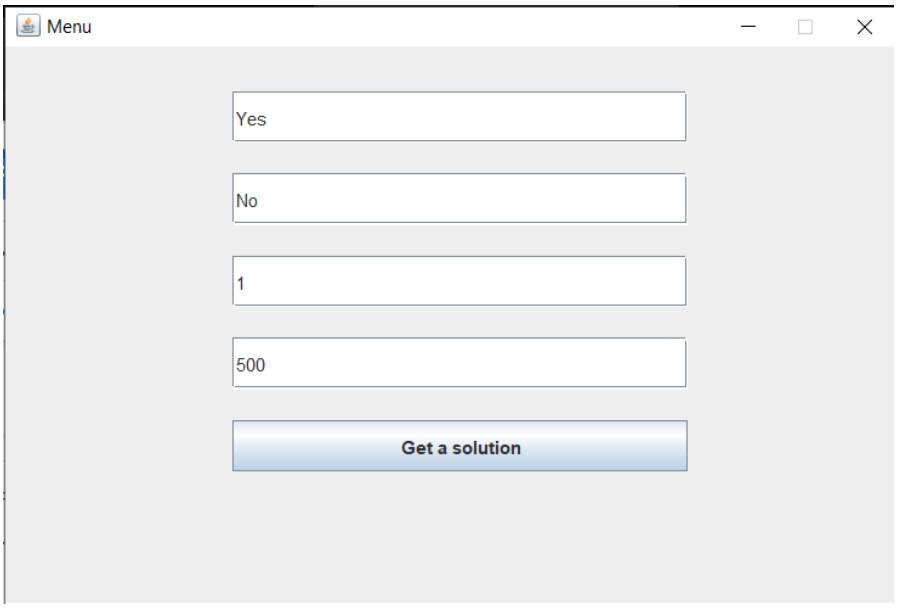

Рис. 6. Пример ответа на вопросы.

В итоге, получаем результат:

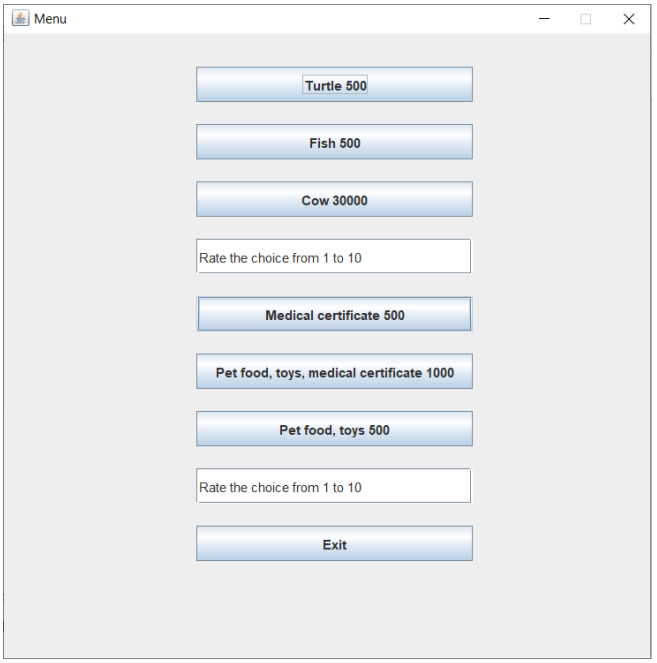

Рис. 7. Результат работы системы.

Примечание: у коровы в БД указано, что её можно заводить людям с аллергией. Попробуем сразу выбрать понравившегося питомца:

Рис. 8. Обработка ошибки выбора.

Теперь же, введём для начала корректную оценку системы, дабы в дальнейшем она учитывала её при своей работе. Нажмём кнопку, получаем:

Рис. 9. Выбор питомца.

Ради интереса попробуем снова выбрать животного тут же. Получаем уведомление:

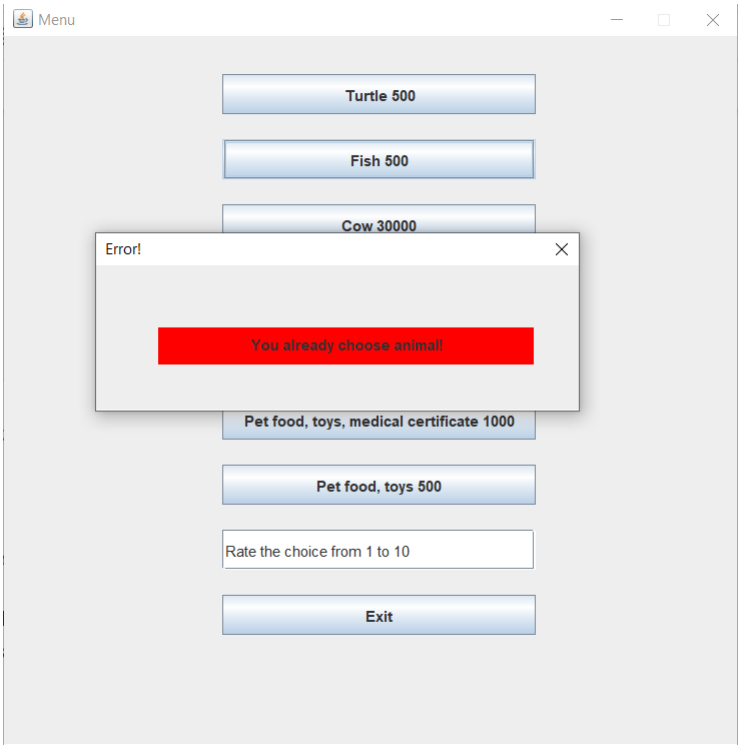

Рис. 10. Обработка повторного выбора.

Выбираем набор, при этом вводя оценку и для него:

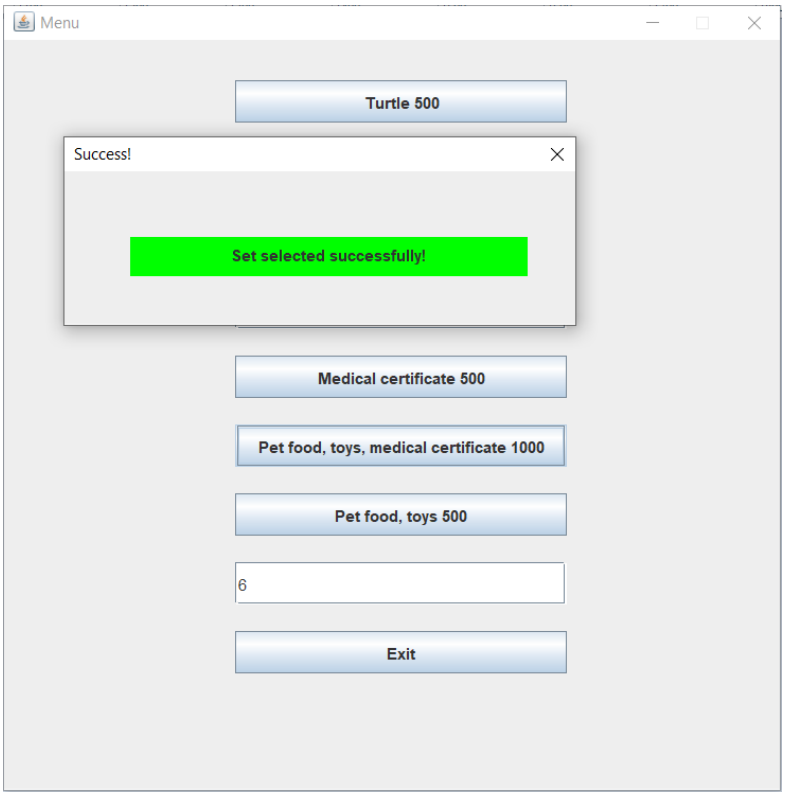

Рис. 11. Выбор набора.

Таким образом, мы протестировали один случай. Поменяем вводные данные и повторим последовательность действий уже сразу с корректным вводом:

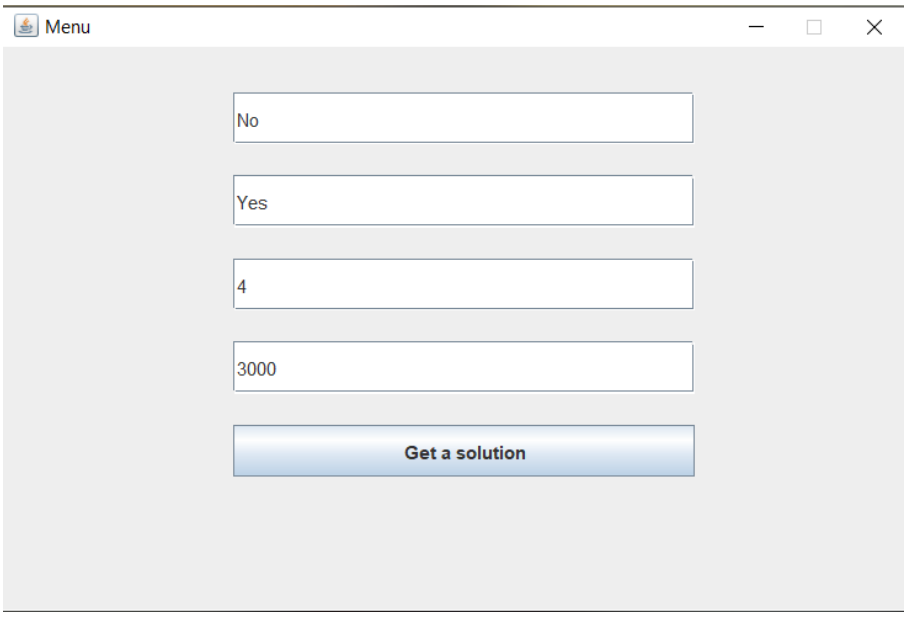

Рис. 12. Входные данные.

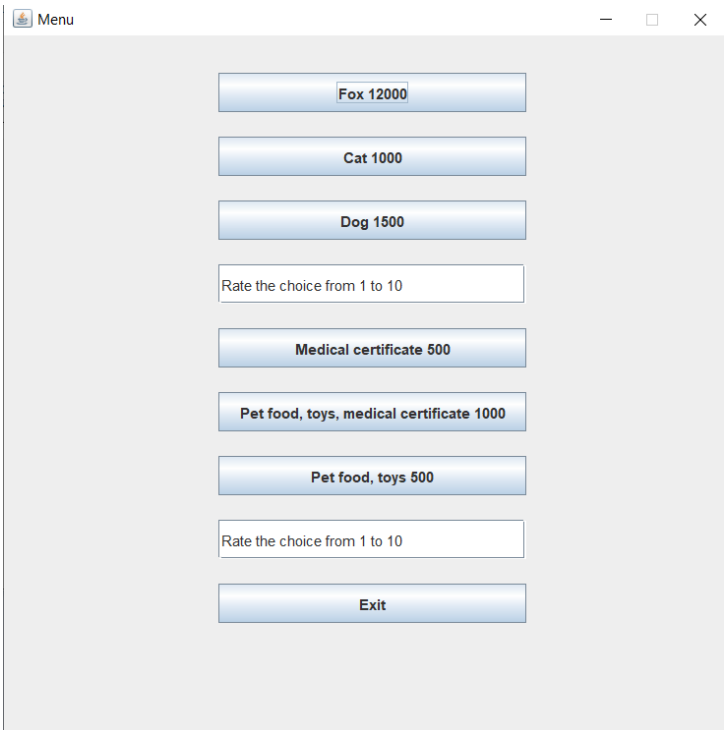

Рис. 13. Результат работы программы.

Теперь же, смоделируем следующую ситуацию: система работает длительное время, пользователи ставят различные отзывы, изменяя базу данных и базу знаний соответственно. Введём те же входные данные, что и в последний раз, и посмотрим на вывод программы:

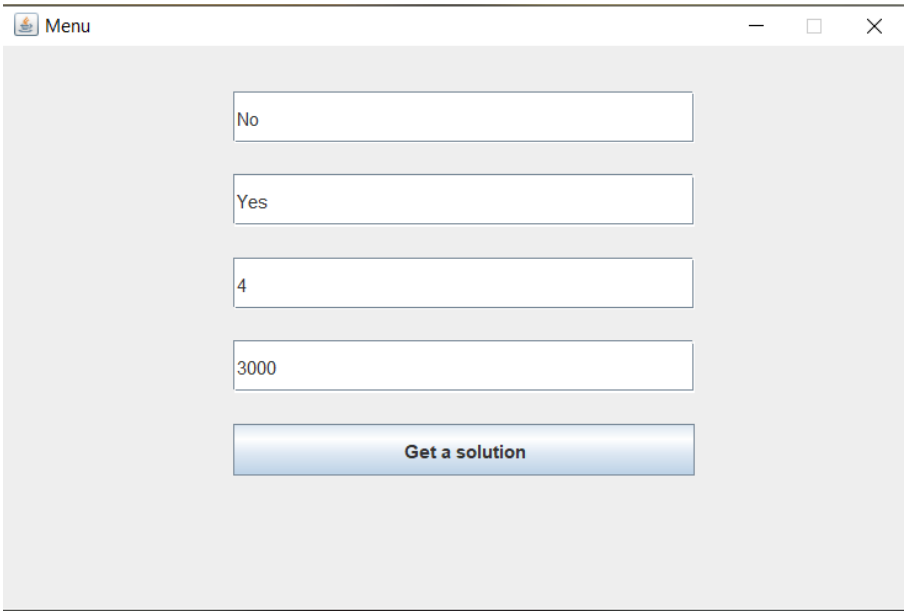

Рис. 14. Входные данные.

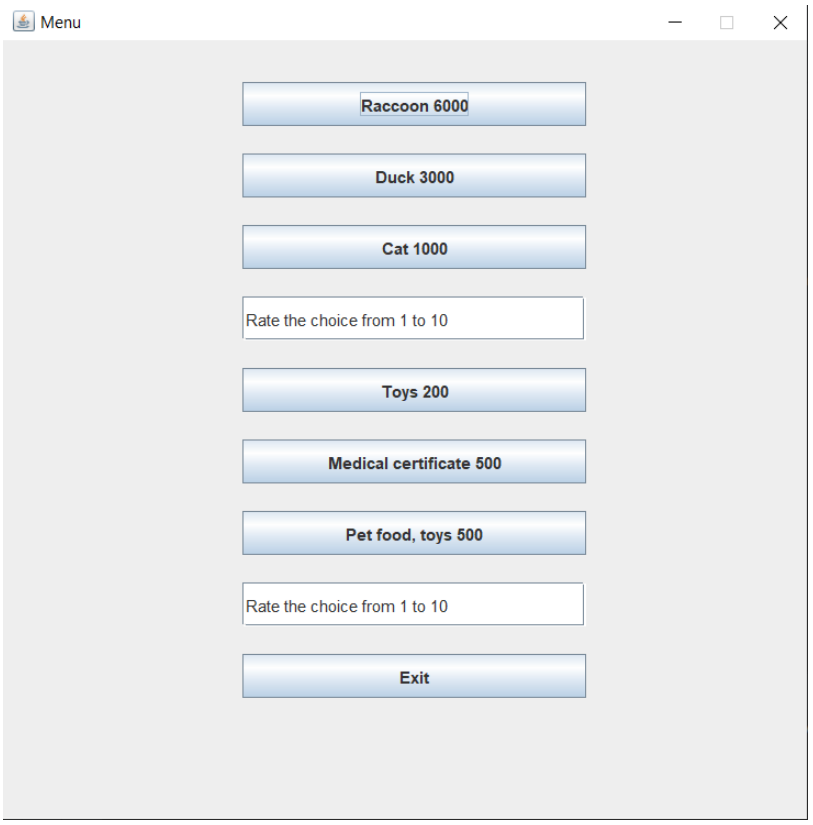

Рис. 15. Результат работы программы.

Таким образом, система является обучаемой и учитывает действия пользователей, что является положительным результатом.

3. Выводы.

В ходе данной лабораторной работы была создана программная реализация нашей системы принятия решений. Таким образом, в результате выполнения лабораторной работы мы на практике ознакомились с разработкой экспертных систем и смогли реализовать все задуманные нами ранее алгоритмы, прежде всего самообучаемость.

4. Листинг программы.

- ***Server.java:***

```
import java.util.ArrayList;

import static java.lang.Math.pow;

public class Server {
   private DataBase dataBase = new DataBase();
   private Decision decision;
   private ArrayList<String> questions = new ArrayList<>();

   public Server() {
       questions.add("Do you have allergies?");
             questions.add("Have you had any experience with handling
animals?");
       questions.add("How much time can you devote to it?");
       questions.add("What price would suit you?");
   }

   public void sendChooseAnimal(int choose, int review) {
       decision.getResultAnimals().get(choose).addSales(review);
       dataBase.update();
   }

   public void sendChooseSet(int choose, int review) {
       decision.getResultSetProducts().get(choose).addSales(review);
       dataBase.update();
   }

      public void updateDecision(String possibleWithAllergies, String
possibleWithoutExperience, int requiredTime, int price) {
                    decision = new Decision(possibleWithAllergies,
possibleWithoutExperience, requiredTime, price);
   }

   public double recalculateRating(Animal animal) {
       if (animal.getName().equals("test")) {
           return -(Double.MAX_VALUE - 5);
       }
       double result = animal.getRating();
                   if ((decision.getHaveAllergies().equals("No")) ||
animal.getPossibleWithAllergies().equals("Yes")) {
                       if (decision.getHaveExperience().equals("No") &&
animal.getPossibleWithoutExperience().equals("No")) {
               result /= 2;
           }
                           result -= pow(animal.getRequiredTime() -
decision.getRequiredTime(), 2);
       } else {
```

```
            return -Integer.MAX_VALUE;
        }
        return result;
    }

    public void calculateDecision() {
        ArrayList<Animal> animals = dataBase.getAnimals();
              Animal testAnimal = new Animal("test", "", "No", "No",
Integer.MAX_VALUE, 0, -100, 0);
        testAnimal.setRating(-Integer.MAX_VALUE);
            Animal firstAnimal = testAnimal, secondAnimal = testAnimal,
thirdAnimal = testAnimal;
        int firstIndex = 0, secondIndex = 0;
        for (int i = 0; i < animals.size(); ++i) {
            Animal currAnimal = animals.get(i);
            double currRating = recalculateRating(currAnimal);
            double rating = firstAnimal.getRating();
            System.out.println(currRating);
            if (rating < currRating) {
                firstAnimal = currAnimal;
                firstIndex = i;
            }
        }
        for (int i = 0; i < animals.size(); ++i) {
            if (i == firstIndex) {
                continue;
            }
            Animal currAnimal = animals.get(i);
            double currRating = recalculateRating(currAnimal);
            if (secondAnimal.getRating() < currRating) {
                secondAnimal = currAnimal;
                secondIndex = i;
            }
        }
        for (int i = 0; i < animals.size(); ++i) {
            if (i == firstIndex || i == secondIndex) {
                continue;
            }
            Animal currAnimal = animals.get(i);
            double currRating = recalculateRating(currAnimal);
            if (thirdAnimal.getRating() < currRating) {
                thirdAnimal = currAnimal;
            }
        }

        ArrayList<SetProducts> setProducts = dataBase.getSets();
        SetProducts testSet = new SetProducts("test", 0, -100, 0);
          SetProducts firstSet = testSet, secondSet = testSet, thirdSet =
testSet;
        firstIndex = 0;
        secondIndex = 0;
        for (int i = 0; i < setProducts.size(); ++i) {
            SetProducts currSet = setProducts.get(i);
```

```
            if (firstSet.getRating() < currSet.getRating()) {
                firstSet = currSet;
                firstIndex = i;
            }
        }
        for (int i = 0; i < setProducts.size(); ++i) {
            if (i == firstIndex) {
                continue;
            }
            SetProducts currSet = setProducts.get(i);
            if (secondSet.getRating() < currSet.getRating()) {
                secondSet = currSet;
                secondIndex = i;
            }
        }
        for (int i = 0; i < setProducts.size(); ++i) {
            if (i == firstIndex || i == secondIndex) {
                continue;
            }
            SetProducts currSet = setProducts.get(i);
            if (thirdSet.getRating() < currSet.getRating()) {
                thirdSet = currSet;
            }
        }
             decision.formSolution(firstAnimal, secondAnimal, thirdAnimal,
firstSet, secondSet, thirdSet);
    }

    public ArrayList<String> getQuestions() {
        return questions;
    }

    public Decision getDecision() {
        return decision;
    }
}
```

- ***MenuFrame.java:***

```
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;

public class MenuFrame extends JFrame {
    private Server server = new Server();
    private final static int WIDTH = 600;
    private final static int HEIGHT = 400;
    private final ArrayList<JTextField> textFields = new ArrayList<>();
    private final JButton sendDecisionButton;

    public MenuFrame() {
        this.setTitle("Menu");
        this.setSize(WIDTH, HEIGHT);
        this.setResizable(false);
```

```
        this.setLayout(null);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        ArrayList<String> questions = server.getQuestions();
        for (int i = 0; i < questions.size(); ++i) {
            textFields.add(new JTextField(questions.get(i)));
        }
        sendDecisionButton = new JButton("Get a solution");
        ;
        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(5, 1, 20, 20));
        for (int i = 0; i < textFields.size(); ++i) {
            panel.add(textFields.get(i));
        }
        panel.add(sendDecisionButton);
        initButtons();
        panel.setBounds(150, 30, 300, 250);
        this.add(panel);
    }

    public void initButtons() {
        sendDecisionButton.addActionListener(e -> {
                                                                    if
(!textFields.get(2).getText().equals(server.getQuestions().get(2))      &&
!textFields.get(3).getText().equals(server.getQuestions().get(3))) {
                        if  ((textFields.get(0).getText().equals("Yes")  ||
textFields.get(0).getText().equals("No")) &&
                             (textFields.get(1).getText().equals("Yes") ||
textFields.get(1).getText().equals("No")) &&
                           (Integer.parseInt(textFields.get(2).getText()) >
0) && (Integer.parseInt(textFields.get(3).getText()) > 0)) {
                        server.updateDecision(textFields.get(0).getText(),
textFields.get(1).getText(),
                              Integer.parseInt(textFields.get(2).getText()),
Integer.parseInt(textFields.get(3).getText()));
                    server.calculateDecision();
                    new ResultFrame().setVisible(true);
                    this.setVisible(false);
                } else {
                     Notification.createNotification(this, "Error!", true,
false, "Please, try again!").setVisible(true);
                }
            } else {
                    Notification.createNotification(this, "Error!", true,
false, "Please, change forms!").setVisible(true);
            }
        });
    }

    public class ResultFrame extends JFrame {
        private final static int WIDTH = 600;
        private final static int HEIGHT = 600;
                private  final  ArrayList<JButton>  buttonsAnimals  =  new
ArrayList<>();
```

```
    private boolean pressedAnimal = false, pressedSet = false;
    private final ArrayList<JButton> buttonsSets = new ArrayList<>();
      private final JTextField reviewAnimal = new JTextField("Rate the
choice from 1 to 10");
       private final JTextField reviewSet = new JTextField("Rate the
choice from 1 to 10");
    private final JButton exitButton = new JButton("Exit");

    public ResultFrame() {
        this.setTitle("Menu");
        this.setSize(WIDTH, HEIGHT);
        this.setResizable(false);
        this.setLayout(null);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(9, 1, 20, 20));

                                        ArrayList<Animal>    animals    =
server.getDecision().getResultAnimals();
        for (int i = 0; i < animals.size(); ++i) {
             buttonsAnimals.add(new JButton(animals.get(i).getName() +
" " + animals.get(i).getPrice()));
        }

                                    ArrayList<SetProducts>   setProducts   =
server.getDecision().getResultSetProducts();
        for (int i = 0; i < setProducts.size(); ++i) {
                                                    buttonsSets.add(new
JButton(setProducts.get(i).getDescription()      +      "      "      +
setProducts.get(i).getPrice()));
        }

        for (int i = 0; i < animals.size(); ++i) {
            panel.add(buttonsAnimals.get(i));
        }
        panel.add(reviewAnimal);
        for (int i = 0; i < setProducts.size(); ++i) {
            panel.add(buttonsSets.get(i));
        }
        panel.add(reviewSet);
        panel.add(exitButton);
        initButtons();
        panel.setBounds(175, 30, 250, 450);
        this.add(panel);
    }

    public void initButtons() {
        for (int i = 0; i < buttonsAnimals.size(); ++i) {
            int finalI = i;
            buttonsAnimals.get(i).addActionListener(e -> {
                    if(!reviewAnimal.getText().equals("Rate the choice
from 1 to 10")) {
                     if (Integer.parseInt(reviewAnimal.getText()) >= 0
```

```
&& Integer.parseInt(reviewAnimal.getText()) <= 10) {
                            if (!pressedAnimal) {
                                                server.sendChooseAnimal(finalI,
Integer.parseInt(reviewAnimal.getText()));
                                pressedAnimal = true;
                                     Notification.createNotification(this,
"Success!", true, true, "Pet selected successfully!").setVisible(true);
                            } else {
                                     Notification.createNotification(this,
"Error!", true, false, "You already choose animal!").setVisible(true);
                            }
                        } else {
                                     Notification.createNotification(this,
"Error!", true, false, "Enter the correct review!").setVisible(true);
                        }
                    } else {
                           Notification.createNotification(this, "Error!",
true, false, "Change forms!").setVisible(true);
                    }
                });
            }
            for (int i = 0; i < buttonsSets.size(); ++i) {
                int finalI = i;
                buttonsSets.get(i).addActionListener(e -> {
                     if(!reviewSet.getText().equals("Rate the choice from 1
to 10")) {
                        if (Integer.parseInt(reviewSet.getText()) >= 1 &&
Integer.parseInt(reviewSet.getText()) <= 10) {
                            if (!pressedSet) {
                                                server.sendChooseSet(finalI,
Integer.parseInt(reviewSet.getText()));
                                pressedSet = true;
                                     Notification.createNotification(this,
"Success!", true, true, "Set selected successfully!").setVisible(true);
                            } else {
                                     Notification.createNotification(this,
"Error!", true, false, "You already choose set!").setVisible(true);
                            }
                        } else {
                                     Notification.createNotification(this,
"Error!", true, false, "Enter the correct review!").setVisible(true);
                        }
                    } else {
                           Notification.createNotification(this, "Error!",
true, false, "Change forms!").setVisible(true);
                    }
                });
            }
            exitButton.addActionListener(e -> {
                this.setVisible(false);
                System.exit(0);
            });
        }
```

```
        }
    }
```

- ***DataBase.java:***

```
import java.io.*;
import java.util.ArrayList;

public class DataBase {
   private final String filepathAnimals = "baseAnimals.txt";
   private final String filepathSets = "baseSets.txt";
   private ArrayList<Animal> animals = new ArrayList<>();
   private ArrayList<SetProducts> sets = new ArrayList<>();

   public DataBase() {
       createBaseAnimals();
       createBaseSets();
   }

   public void createBaseAnimals() {
       try {
                       BufferedReader bufferedReader = new BufferedReader(new
FileReader(filepathAnimals));
           String buffer;
           int counter = 1;
           String name = "";
           String description = "";
           String possibleWithAllergies = "";
           String possibleWithoutExperience = "";
           int requiredTime = 0;
           int price = 0;
           int numberOfSales = 0;
           double averageReview = 0;
           while ((buffer = bufferedReader.readLine()) != null) {
               if (buffer.equals("End")) {
                                   animals.add(new Animal(name, description,
possibleWithAllergies, possibleWithoutExperience, requiredTime, price,
numberOfSales, averageReview));
                   counter = 1;
                   continue;
               }
               switch (counter) {
                   case 1:
                       name = buffer;
                       break;
                   case 2:
                       description = buffer;
                       break;
                   case 3:
                       possibleWithAllergies = buffer;
                       break;
                   case 4:
                       possibleWithoutExperience = buffer;
```

```
                    break;
                case 5:
                    requiredTime = Integer.parseInt(buffer);
                    break;
                case 6:
                    price = Integer.parseInt(buffer);
                    break;
                case 7:
                    numberOfSales = Integer.parseInt(buffer);
                    break;
                case 8:
                    averageReview = Double.parseDouble(buffer);
            }
            counter++;
        }
    } catch (IOException exception) {
        exception.printStackTrace();
    }
}

public void createBaseSets() {
    try {
                BufferedReader bufferedReader = new BufferedReader(new
FileReader(filepathSets));
        String buffer;
        int counter = 1;
        String description = "";
        int price = 0;
        int numberOfSales = 0;
        double averageReview = 0;
        while ((buffer = bufferedReader.readLine()) != null) {
            if (buffer.equals("End")) {
                  sets.add(new SetProducts(description, price, numberOfSales,
averageReview));
                counter = 1;
                continue;
            }
            switch (counter) {
                case 1:
                    description = buffer;
                    break;
                case 2:
                    price = Integer.parseInt(buffer);
                    break;
                case 3:
                    numberOfSales = Integer.parseInt(buffer);
                    break;
                case 4:
                    averageReview = Double.parseDouble(buffer);
                    break;
            }
            counter++;
        }
```

```
        } catch (IOException exception) {
            exception.printStackTrace();
        }
    }

    public void update() {
        updateAnimals();
        updateSets();
    }

    public void updateAnimals() {
        try {
                        BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter(filepathAnimals));
            for (int i = 0; i < animals.size(); ++i) {
                Animal currentAnimal = animals.get(i);
                bufferedWriter.write(currentAnimal.getName() + "\n");
                bufferedWriter.flush();
                bufferedWriter.write(currentAnimal.getDescription() + "\n");
                bufferedWriter.flush();
                  bufferedWriter.write(currentAnimal.getPossibleWithAllergies() +
"\n");
                bufferedWriter.flush();

bufferedWriter.write(currentAnimal.getPossibleWithoutExperience() + "\n");
                bufferedWriter.flush();
                bufferedWriter.write(currentAnimal.getRequiredTime() + "\n");
                bufferedWriter.flush();
                bufferedWriter.write(currentAnimal.getPrice() + "\n");
                bufferedWriter.flush();
                bufferedWriter.write(currentAnimal.getNumberOfSales() + "\n");
                bufferedWriter.flush();
                bufferedWriter.write(currentAnimal.getAverageReview() + "\n");
                bufferedWriter.flush();
                bufferedWriter.write("End\n");
                bufferedWriter.flush();
            }
        } catch (IOException exception) {
            exception.printStackTrace();
        }
    }

    public void updateSets() {
        try {
                        BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter(filepathSets));
            for (int i = 0; i < sets.size(); ++i) {
                SetProducts currentSet = sets.get(i);
                bufferedWriter.write(currentSet.getDescription() + "\n");
                bufferedWriter.flush();
                bufferedWriter.write(currentSet.getPrice() + "\n");
                bufferedWriter.flush();
                bufferedWriter.write(currentSet.getNumberOfSales() + "\n");
```

```
                bufferedWriter.flush();
                bufferedWriter.write(currentSet.getAverageReview() + "\n");
                bufferedWriter.flush();
                bufferedWriter.write("End\n");
                bufferedWriter.flush();
            }
        } catch (IOException exception) {
            exception.printStackTrace();
        }
    }

    public ArrayList<Animal> getAnimals() {
        return animals;
    }

    public ArrayList<SetProducts> getSets() {
        return sets;
    }

}
```