**Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования** **«Новгородский государственный университет имени Ярослава Мудрого»**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра ИТиС

«Поисковая система на основе фреймовой модели представления знаний»

Лабораторная работа №3

По дисциплине: Системы искусственного интеллекта

Отчет

Выполнил: студент гр. 9091

\_\_\_\_\_\_\_ \_\_\_\_Мирошкин О.А.

Проверил: Преподаватель

\_\_\_\_\_\_\_\_\_\_\_Михайлов Д.В.

Великий Новгород

2022

1. **Формулировка цели и задач**

Рассмотреть представление знаний фреймами и соответствующие способы управления выводом.

1. **Модель представления знаний**

Предметная область – Материал для пошива одежды. Данные были взяты из лабораторной работы #2 с добавлением новых данных, чтобы разнообразить их различными типами.

Фреймом называется структура для описания понятия или ситуации, состоящая из характеристик этой ситуации и их значений.

Особенность фреймовых моделей представления знаний состоит в том, что все понятия, описываемые в каждом из узлов модели, определяются набором атрибутов и их значениями, которые содержатся в слотах фрейма.

Слот — это атрибут, который связывается с узлом в системе, основанной на фреймах. Он является основной составляющей фрейма. Он имеет имя слота и значение слота.

С каждым слотом может быть связана одна или несколько процедур, которые выполняются, когда изменяются значения слотов. Чаще всего со слотами связываются процедуры:

ЕСЛИ-ДОБАВЛЕНО (IF-ADDED) — выполняется в том случае, когда новая информация помещается в слот;

Эта процедура следит за изменением информации, приписываемой данному узлу, и проверяет, что при изменении тех или иных значений производятся соответствующие действия.

Фреймовые системы позволяют использовать объектно-ориентированный подход к представлению знаний.

Демон “Проверка на >=0” является демоном IF-ADDED.

Таблица . Фрейм "Ткань"

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя слота** | **Указатель наследования** | **Тип данных** | **Значение слота** | **Демон** |
| Цена за метр | U(Unique) | Real | 0 | Проверка на >= 0 |
| Качество | U(Unique) | Integer | 0 | Проверка на >= 0 |
| Аллергия | U(Unique) | Integer | 0 | Проверка на >= 0 |
| Поглощение влаги | U(Unique) | Integer | 0 | Проверка на >= 0 |
| Воздухопроницаемость | U(Unique) | Integer | 0 | Проверка на >= 0 |
| Терморегуляция | U(Unique) | Integer | 0 | Проверка на >= 0 |
| Возможно получить с доставкой | U(Unique) | Boolean | True |  |
| Объявление натурального | S(Same) | Lisp | “создана новая натуральная ткань!” | Проверка на >= 0 |

Таблица 2. Фрейм "Натуральный"

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя слота** | **Указатель наследования** | **Тип данных** | **Значение слота** | **Демон** |
| Сжатие при стирке | U(Unique) | Integer | 0 | Проверка на >= 0 |

Таблица 3. Фрейм "Искусственный"

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя слота** | **Указатель наследования** | **Тип данных** | **Значение слота** | **Демон** |
| Электризуемость | U(Unique) | Integer | 0 | Проверка на >= 0 |

Таблица 4. Фрейм "Хлопок"

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя слота** | **Указатель наследования** | **Тип данных** | **Значение слота** | **Демон** |
| Цена за метр | U(Unique) | Real | 1137 | Проверка на >= 0 |
| Качество | U(Unique) | Integer | 8 | Проверка на >= 0 |
| Аллергия | U(Unique) | Integer | 2 | Проверка на >= 0 |
| Поглощение влаги | U(Unique) | Integer | 3 | Проверка на >= 0 |
| Воздухопроницаемость | U(Unique) | Integer | 3 | Проверка на >= 0 |
| Терморегуляция | U(Unique) | Integer | 3 | Проверка на >= 0 |
| Возможно получить с доставкой | U(Unique) | Boolean | True |  |
| Объявление натурального | S(Same) | Lisp | “создана новая натуральная ткань!” |  |
| Сжатие при стирке | U(Unique) | Integer | 6 | Проверка на >= 0 |

Таблица 5. Фрейм "Бамбук"

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя слота** | **Указатель наследования** | **Тип данных** | **Значение слота** | **Демон** |
| Цена за метр | U(Unique) | Real | 600 | Проверка на >= 0 |
| Качество | U(Unique) | Integer | 4 | Проверка на >= 0 |
| Аллергия | U(Unique) | Integer | 3 | Проверка на >= 0 |
| Поглощение влаги | U(Unique) | Integer | 2 | Проверка на >= 0 |
| Воздухопроницаемость | U(Unique) | Integer | 5 | Проверка на >= 0 |
| Терморегуляция | U(Unique) | Integer | 2 | Проверка на >= 0 |
| Возможно получить с доставкой | U(Unique) | Boolean | True |  |
| Объявление натурального | S(Same) | Lisp | “создана новая натуральная ткань!” |  |
| Электризуемость | U(Unique) | Integer | 4 | Проверка на >= 0 |

Таблица 6. Фрейм "Акрил"

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя слота** | **Указатель наследования** | **Тип данных** | **Значение слота** | **Демон** |
| Цена за метр | U(Unique) | Real | 400 | Проверка на >= 0 |
| Качество | U(Unique) | Integer | 2 | Проверка на >= 0 |
| Аллергия | U(Unique) | Integer | 2 | Проверка на >= 0 |
| Поглощение влаги | U(Unique) | Integer | 0 | Проверка на >= 0 |
| Воздухопроницаемость | U(Unique) | Integer | 1 | Проверка на >= 0 |
| Терморегуляция | U(Unique) | Integer | 2 | Проверка на >= 0 |
| Возможно получить с доставкой | U(Unique) | Boolean | True |  |
| Объявление натурального | S(Same) | Lisp | “создана новая натуральная ткань!” |  |

1. **Механизм управления вводом**

Фреймовые системы позволяют использовать объектно-ориентированный подход к представлению знаний. Эта структура базируется на отношениях IS-А между фреймами, описывающими некую модель данных.

При этом наши представления и знания об объекте "Ткань" детализируем рядом понятий, которые определим через набор атрибутов и возможными типы их значений. Все атрибуты разместим в слотах фрейма.

Таким образом, на самом верхнем уровне иерархии определен фрейм “Ткань”. В данном случае ткань разделяется на 3 типа: ”Натуральный”, “Искусственный”, “Синтетический”, которые определены в качестве соответствующих дочерних фреймов.

Фреймы, описывающие различные объекты называются шаблонами, а фреймы верхнего уровня, использующиеся для представления этих шаблонов, называются фреймами класса.

Ткань <= IS\_A <= Натуральный

Ткань <= IS\_A <= Искусственный

Ткань <= IS\_A <= Синтетический

Натуральный <= IS\_A <= Хлопок

Искусственный <= IS\_A <= Бамбук

Синтетический <= IS\_A <= Акрил

1. **Описание механизма поиска по запросу в поисковой системе**

Реализовано несколько способов поиска по базе данных.

**Найти фрейм по имени:** при выполнении система запрашивает имя фрейма для поиска и выводит результат.

Пример:

*Ввод: Акрил*

*Ответ: Фрейм Акрил создан.*

*Ввод: Джордан*

*Ответ: Фрейм Джордан не создан.*

**Найти фрейм по имени слота:** при выполнении система запрашивает имя слота и выводит названия фреймов, в которых содержится такой слот.

*Ввод: Электризуемость*

*Ответ: Найденные фреймы: Искусственный, Бамбук*

**Найти фрейм по значению слота:** при выполнении система запрашивает значение слота и выводит фреймы, в которых есть слот с таким значением.

*Ввод: 1137*

*Ответ: Найденные фреймы: Хлопок*

1. **Тестовые наборы для всех функций преобразования над знаниями в системе**

Создание фрейма

*Ввод: Джордан*

*Вывод: Фрейм Джордан успешно создан!*

Попытка создания существующего фрейма

*Ввод: Натуральный*

*Вывод: Фрейм с именем Натуральный уже создан! Введите другое имя.*

Создание слота

*Введите имя слота*:

*ТестСлот*

*Введите тип указателя наследования [Unique, Same]:*

*Unique*

*Введите тип данных слота [Real, Integer, Boolean, Text, Lisp]:*

*Integer*

*Введите значение слота (тип Integer):*

*5*

Попытка создания существующего слота

*ТестСлот*

*Слот с именем ТестСлот уже создан во фрейме Акрил! Введите другое имя.*

Редактирование значения существующего слота

*Введите имя слота:*

*ТестСлот*

*Введите значение слота (тип Integer):*

*9*

*Успешно изменили значение слота ТестСлот на 9*

Установка наследования фрейма (открыт фрейм Акрил)

*Введите имя фрейма родителя:*

*Ткань*

*Успешно установлено наследование Акрил<-Ткань!*

Попытка установления наследования в самого себя

*Введите имя фрейма родителя:*

*Акрил*

*Фрейм не может наследоваться от себя! Попробуйте ещё раз.*

Удаление фрейма (открыт фрейм Акрил)

*Фрейм Акрил успешно удалён!*

Ввод числа, не удовлетворяющего критерию присоединённого демона

*Введите имя слота:*

*Качество*

*Введите значение слота (тип Integer):*

*-5*

*Число -5 не удовлетворяет условию >= 0*

*Неправильный ввод! Попробуйте ещё раз.*

Отрабатывание процедуры “Объявление натурального”

*Введите имя фрейма родителя:*

*Натуральный*

*Успешно установлено наследование Натуральный<-Хлопок!*

*Ткань: создана новая натуральная ткань!*

1. **Исходный код программы**

***FrameDatabase.json***

[{"Name":"Ткань","ParentName":null,"Slots":[{"Name":"Цена за метр","InheritType":0,"DataType":0,"DataValue":0.0},{"Name":"Качество","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Аллергия","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Поглощение влаги","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Воздухопроницаемость","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Терморегуляция","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Возможно получить с доставкой","InheritType":0,"DataType":2,"DataValue":true},{"Name":"Объявление натурального","InheritType":0,"DataType":5,"DataValue":{"PrintMessage":"создана новая натуральная ткань!","OwnerName":"Ткань","LispType":3,"ExecSlotName":null,"ExecSlotValue":null,"ExecInheritorName":"Натуральный"}}]},{"Name":"Натуральный","ParentName":null,"Slots":[{"Name":"Сжатие при стирке","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Цена за метр","InheritType":0,"DataType":0,"DataValue":0.0},{"Name":"Качество","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Аллергия","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Поглощение влаги","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Воздухопроницаемость","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Терморегуляция","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Возможно получить с доставкой","InheritType":0,"DataType":2,"DataValue":true},{"Name":"Объявление натурального","InheritType":0,"DataType":5,"DataValue":{"PrintMessage":"создана новая натуральная ткань!","OwnerName":"Ткань","LispType":3,"ExecSlotName":null,"ExecSlotValue":null,"ExecInheritorName":"Натуральный"}}]},{"Name":"Хлопок","ParentName":null,"Slots":[{"Name":"Сжатие при стирке","InheritType":0,"DataType":1,"DataValue":6},{"Name":"Цена за метр","InheritType":0,"DataType":0,"DataValue":1137},{"Name":"Качество","InheritType":0,"DataType":1,"DataValue":8},{"Name":"Аллергия","InheritType":0,"DataType":1,"DataValue":2},{"Name":"Поглощение влаги","InheritType":0,"DataType":1,"DataValue":3},{"Name":"Воздухопроницаемость","InheritType":0,"DataType":1,"DataValue":3},{"Name":"Терморегуляция","InheritType":0,"DataType":1,"DataValue":3},{"Name":"Возможно получить с доставкой","InheritType":0,"DataType":2,"DataValue":true},{"Name":"Объявление натурального","InheritType":0,"DataType":5,"DataValue":{"PrintMessage":"создана новая натуральная ткань!","OwnerName":"Ткань","LispType":3,"ExecSlotName":null,"ExecSlotValue":null,"ExecInheritorName":"Натуральный"}}]},{"Name":"Искусственный","ParentName":"Ткань","Slots":[{"Name":"Электризуемость","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Цена за метр","InheritType":0,"DataType":0,"DataValue":0.0},{"Name":"Качество","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Аллергия","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Поглощение влаги","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Воздухопроницаемость","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Терморегуляция","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Возможно получить с доставкой","InheritType":0,"DataType":2,"DataValue":true},{"Name":"Объявление натурального","InheritType":0,"DataType":5,"DataValue":{"PrintMessage":"создана новая натуральная ткань!","OwnerName":"Ткань","LispType":3,"ExecSlotName":null,"ExecSlotValue":null,"ExecInheritorName":"Натуральный"}}]},{"Name":"Бамбук","ParentName":"Искусственный","Slots":[{"Name":"Электризуемость","InheritType":0,"DataType":1,"DataValue":4},{"Name":"Цена за метр","InheritType":0,"DataType":0,"DataValue":600},{"Name":"Качество","InheritType":0,"DataType":1,"DataValue":4},{"Name":"Аллергия","InheritType":0,"DataType":1,"DataValue":3},{"Name":"Поглощение влаги","InheritType":0,"DataType":1,"DataValue":2},{"Name":"Воздухопроницаемость","InheritType":0,"DataType":1,"DataValue":5},{"Name":"Терморегуляция","InheritType":0,"DataType":1,"DataValue":2},{"Name":"Возможно получить с доставкой","InheritType":0,"DataType":2,"DataValue":true},{"Name":"Объявление натурального","InheritType":0,"DataType":5,"DataValue":{"PrintMessage":"создана новая натуральная ткань!","OwnerName":"Ткань","LispType":3,"ExecSlotName":null,"ExecSlotValue":null,"ExecInheritorName":"Натуральный"}}]},{"Name":"Синтетический","ParentName":"Ткань","Slots":[{"Name":"Цена за метр","InheritType":0,"DataType":0,"DataValue":0.0},{"Name":"Качество","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Аллергия","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Поглощение влаги","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Воздухопроницаемость","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Терморегуляция","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Возможно получить с доставкой","InheritType":0,"DataType":2,"DataValue":true},{"Name":"Объявление натурального","InheritType":0,"DataType":5,"DataValue":{"PrintMessage":"создана новая натуральная ткань!","OwnerName":"Ткань","LispType":3,"ExecSlotName":null,"ExecSlotValue":null,"ExecInheritorName":"Натуральный"}}]},{"Name":"Акрил","ParentName":"Синтетический","Slots":[{"Name":"Цена за метр","InheritType":0,"DataType":0,"DataValue":400},{"Name":"Качество","InheritType":0,"DataType":1,"DataValue":2},{"Name":"Аллергия","InheritType":0,"DataType":1,"DataValue":2},{"Name":"Поглощение влаги","InheritType":0,"DataType":1,"DataValue":0},{"Name":"Воздухопроницаемость","InheritType":0,"DataType":1,"DataValue":1},{"Name":"Терморегуляция","InheritType":0,"DataType":1,"DataValue":2},{"Name":"Возможно получить с доставкой","InheritType":0,"DataType":2,"DataValue":true},{"Name":"Объявление натурального","InheritType":0,"DataType":5,"DataValue":{"PrintMessage":"создана новая натуральная ткань!","OwnerName":"Ткань","LispType":3,"ExecSlotName":null,"ExecSlotValue":null,"ExecInheritorName":"Натуральный"}}]}]

***Файл UserInterface.cs***

using FrameKnowledgeModel.Data;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace FrameKnowledgeModel

{

public enum CommandType

{

ExitApplication, CreateFrame, OpenFrame,

FindFrameByName, FindFrameBySlotName, FindFrameBySlotValue, PrintAllFrames,

SaveToDisk,

DeleteFrame, CreateSlot, RedactSlotValue, DeleteSlot, SetInheritance, CloseFrame

}

public class UserInterface

{

private const CommandType OpenedFrameCommandsStart = CommandType.DeleteFrame;

private readonly static Dictionary<CommandType, string> \_commandTranslations = new Dictionary<CommandType, string>

{

{ CommandType.ExitApplication, "Завершить программу" },

{ CommandType.CreateFrame, "Создать фрейм" },

{ CommandType.OpenFrame, "Открыть фрейм" },

{ CommandType.FindFrameByName, "Найти фрейм по имени" },

{ CommandType.FindFrameBySlotName, "Найти фрейм(ы) по имени слота" },

{ CommandType.FindFrameBySlotValue, "Найти фрейм(ы) по значению слота" },

{ CommandType.PrintAllFrames, "Вывести все фреймы" },

{ CommandType.SaveToDisk, "Сохранить данные на диск" },

{ CommandType.DeleteFrame, "Удалить фрейм" },

{ CommandType.CreateSlot, "Создать слот" },

{ CommandType.RedactSlotValue, "Отредактировать значение слота" },

{ CommandType.DeleteSlot, "Удалить слот" },

{ CommandType.SetInheritance, "Установить наследование" },

{ CommandType.CloseFrame, "Закрыть фрейм" }

};

private Frame \_openedFrame;

public CommandType RequestCommand(Frame openedFrame)

{

\_openedFrame = openedFrame;

if (openedFrame != null)

{

PrintNewLine();

Console.WriteLine("Открытый фрейм: ");

PrintFrame(openedFrame);

PrintNewLine();

}

Console.WriteLine("Что вы хотите сделать?\n");

IEnumerable<CommandType> commands = Enum.GetValues(typeof(CommandType)).Cast<CommandType>();

foreach (var command in commands)

{

if (command == OpenedFrameCommandsStart)

{

if (openedFrame == null)

{

Console.WriteLine("Откройте фрейм чтобы увидеть больше команд.");

break;

}

}

Console.WriteLine($"{(int)command}: {\_commandTranslations[command]}");

}

PrintNewLine();

string reply = Console.ReadLine();

if (!int.TryParse(reply, out int commandId) || !Enum.IsDefined(typeof(CommandType), commandId))

{

Console.WriteLine("Неправильный ввод! Попробуйте ещё раз.");

return RequestCommand(openedFrame);

}

return (CommandType)commandId;

}

public void RequestIsFrameCreatedCheck(IEnumerable<string> createdFrameNames)

{

RequestFrameName(createdFrameNames, "", out string frameName, out bool alreadyCreated);

string applyResult = alreadyCreated ? "" : " не";

Console.WriteLine($"Фрейм {frameName}{applyResult} создан.");

PrintNewLine();

}

public string RequestCreatedFrameName(IEnumerable<string> createdFrameNames, string additionalRequestMessage = "")

{

RequestFrameName(createdFrameNames, additionalRequestMessage, out string frameName, out bool alreadyCreated);

if (!alreadyCreated)

{

Console.WriteLine($"Не найден фрейм с именем {frameName}! Введите другое имя.");

return RequestCreatedFrameName(createdFrameNames);

}

return frameName;

}

public string RequestFrameCreationName(IEnumerable<string> createdFrameNames, string additionalRequestMessage = "")

{

RequestFrameName(createdFrameNames, additionalRequestMessage, out string frameName, out bool alreadyCreated);

if (alreadyCreated)

{

Console.WriteLine($"Фрейм с именем {frameName} уже создан! Введите другое имя.");

return RequestFrameCreationName(createdFrameNames);

}

return frameName;

}

public void PrintSuccessDelete(string frameName)

{

Console.WriteLine($"Фрейм {frameName} успешно удалён!");

}

public void PrintFoundFrames(IEnumerable<Frame> foundFrames)

{

bool anyFrameFound = foundFrames.Any();

if (anyFrameFound)

{

Console.WriteLine($"Найденные фреймы: {foundFrames.Select(frame => frame.Name).ToSingleStr()}");

}

else

{

Console.WriteLine("Фреймы, удовлетворяющие условию, не найдены.");

}

PrintNewLine();

}

public void PrintSuccessFrameCreation(string frameName) =>

Console.WriteLine($"Фрейм {frameName} успешно создан!");

public void PrintSuccessInhertaince(string childName, string parentName) =>

Console.WriteLine($"Успешно установлено наследование {childName}<-{parentName}!");

public Slot RequestSlotCreation(Frame openedFrame)

{

IEnumerable<string> createdSlotNames = openedFrame.Slots.Select(slot => slot.Name);

string slotName = RequestSlotCreationName(createdSlotNames, openedFrame.Name);

InheritType inheritType = Extenstions.RequestEnumValue<InheritType>(

"Введите тип указателя наследования",

"Неправильный ввод указателя наследования! Попробуйте ещё раз.");

DataType dataType = RequestSlotValueType();

object slotValue = RequestSlotValue(dataType);

return new Slot(slotName, inheritType, dataType, slotValue);

}

public void PrintAllFrames(IEnumerable<Frame> frames)

{

PrintNewLine();

foreach (var frame in frames)

{

PrintFrame(frame);

PrintNewLine();

}

}

public void PrintFrame(Frame frame)

{

Console.ForegroundColor = ConsoleColor.Cyan;

Console.WriteLine($"Имя фрейма: {frame.Name}");

foreach (Slot slot in frame.Slots)

{

Console.ForegroundColor = ConsoleColor.Yellow;

Console.Write(string.Format("Имя слота: {0,-33}", slot.Name));

Console.Write($"Указатель наследования: {slot.InheritType}\t");

Console.Write($"Указатель типа данных: {slot.DataType}\t");

Console.Write($"Значение слота: {slot.DataValue}\t");

PrintNewLine();

}

Console.ResetColor();

PrintNewLine();

}

public void PrintFrameClosed(string frameName) =>

Console.WriteLine($"Фейм {frameName} закрыт.");

public void RequestSlotEditing(Frame frameToEdit)

{

IReadOnlySlot slotToEdit = FindSlotByName(frameToEdit);

object slotValue = RequestSlotValue(slotToEdit.DataType);

frameToEdit.EditSlotValue(slotToEdit, slotValue);

Console.WriteLine($"Успешно изменили значение слота {slotToEdit.Name} на {slotToEdit.DataValue}");

}

public void PrintSuccessDiskSave()

{

PrintNewLine();

Console.WriteLine("Данные успешно сохранены!");

}

public void PrintNewLine() => Console.WriteLine(string.Empty);

private static IReadOnlySlot FindSlotByName(Frame frame)

{

Console.WriteLine($"Введите имя слота: ");

string slotName = Console.ReadLine();

IReadOnlySlot slot = frame.Slots.FirstOrDefault(slot => slot.Name == slotName);

while (slot == default)

{

Console.WriteLine($"Слот с именем {slotName} не найден во фрейме {frame.Name}! Введите другое имя.");

slotName = Console.ReadLine();

slot = frame.Slots.FirstOrDefault(slot => slot.Name == slotName);

}

return slot;

}

public string RequestSlotName(string additionalMessage = "")

{

Console.WriteLine($"Введите имя слота: {additionalMessage}");

string slotName = Console.ReadLine();

return slotName;

}

private string RequestSlotCreationName(IEnumerable<string> createdSlotNames, string frameName)

{

string slotName = RequestSlotName();

while (createdSlotNames.Contains(slotName))

{

Console.WriteLine($"Слот с именем {slotName} уже создан во фрейме {frameName}! Введите другое имя.");

slotName = Console.ReadLine();

}

return slotName;

}

public string RequestAnySlotValue()

{

Console.WriteLine($"Введите значение слота: ");

return Console.ReadLine().Trim().ToLower();

}

public object RequestSlotValue(DataType dataType)

{

object slotValue = null;

while (slotValue == null)

{

Console.WriteLine($"Введите значение слота (тип {dataType}): ");

if (dataType == DataType.Lisp)

{

Console.WriteLine($"Поддерживается функция print. Введите текст для вывода функцией: ");

}

string valueReply = Console.ReadLine().Trim().ToLower();

switch (dataType)

{

case DataType.Boolean:

if (bool.TryParse(valueReply, out bool boolValue))

slotValue = boolValue;

break;

case DataType.Integer:

if (int.TryParse(valueReply, out int intValue) && CheckNumberValid(intValue))

slotValue = intValue;

break;

case DataType.Real:

if (double.TryParse(valueReply, out double doubleValue) && CheckNumberValid((int)doubleValue))

slotValue = doubleValue;

break;

case DataType.Text:

slotValue = valueReply;

break;

case DataType.Frame:

throw new NotImplementedException();

case DataType.Lisp:

if (TryCreateInputLisp(out Lisp lisp, valueReply))

slotValue = lisp;

break;

default:

throw new NotImplementedException();

}

if (slotValue != null) break;

Console.WriteLine("Неправильный ввод! Попробуйте ещё раз.");

}

return slotValue;

}

public static void RequestFrameName(IEnumerable<string> createdFrameNames, string additionalRequestMessage,

out string frameName, out bool alreadyCreated)

{

Console.WriteLine($"Введите имя фрейма {additionalRequestMessage}: ");

frameName = Console.ReadLine();

alreadyCreated = createdFrameNames.Contains(frameName);

}

public void PrintExecutedLisp(Lisp lisp) => Console.WriteLine($"{lisp.OwnerName}: {lisp.PrintMessage}");

private bool TryCreateInputLisp(out Lisp lisp, string lispPrintMessage)

{

lisp = null;

Console.WriteLine($"Введите когда должна сработать функция: \n" +

$"{(int)LispType.ThisFrame}: При добавлении слота в этот фрейм\n" +

$"{(int)LispType.SlotName}: При добавлении слота с особым именем в любой фрейм\n" +

$"{(int)LispType.SlotValue}: При добавлении слота со значением в любой фрейм\n" +

$"{(int)LispType.Inherit}: При добавлении слота-наследника");

string typeInput = Console.ReadLine();

if (!Enum.TryParse(typeInput, out LispType lispType))

{

return false;

}

string slotSearchName = null;

object slotSearchValue = null;

string searchInheritorName = null;

switch (lispType)

{

case LispType.SlotName:

Console.WriteLine("Введите имя слота: ");

slotSearchName = Console.ReadLine();

break;

case LispType.SlotValue:

DataType typeToPrint = RequestSlotValueType(DataType.Lisp, DataType.Frame);

slotSearchValue = RequestSlotValue(typeToPrint);

break;

case LispType.Inherit:

Console.WriteLine("Введите имя наследника: ");

searchInheritorName = Console.ReadLine();

break;

}

lisp = new Lisp(\_openedFrame.Name, lispPrintMessage, lispType,

slotSearchName, slotSearchValue, searchInheritorName);

return true;

}

private DataType RequestSlotValueType(params DataType[] toExclude)

{

const DataType notReleased = DataType.Frame;

return Extenstions.RequestEnumValue(

"Введите тип данных слота",

"Неправильный ввод типа данных! Попробуйте ещё раз.",

toExclude.Append(notReleased));

}

private bool CheckNumberValid(int number)

{

bool valid = number >= 0;

if (!valid)

{

Console.WriteLine($"Число {number} не удовлетворяет условию >= 0");

}

return valid;

}

}

}

***Файл Program.cs***

using FrameKnowledgeModel.Data;

using System;

using System.Collections.Generic;

using System.Linq;

namespace FrameKnowledgeModel

{

class Program

{

private const string FrameDatabasePath = "FrameDatabase.json";

private static List<Frame> \_createdFrames;

private static Frame \_openedFrame;

static void Main(string[] args)

{

UserInterface userInterface = new UserInterface();

FrameFinder frameFinder = new FrameFinder(userInterface);

FrameSerializer frameSerializer = new FrameSerializer(FrameDatabasePath);

\_createdFrames = frameSerializer.Load();

\_createdFrames ??= new List<Frame>();

frameFinder.SetStartingFrameInheritance(\_createdFrames);

IEnumerable<Lisp> availableLisps = \_createdFrames

.SelectMany(frame => frame.Slots)

.Where(slot => slot.DataType == DataType.Lisp)

.Select(slot => slot.DataValue as Lisp);

IEnumerable<Lisp> inheritTypeLisps = availableLisps.Where(lisp => lisp.LispType == LispType.Inherit);

IEnumerable<string> frameNames = \_createdFrames.Select(frame => frame.Name);

while (true)

{

CommandType command = userInterface.RequestCommand(\_openedFrame);

switch (command)

{

case CommandType.ExitApplication:

ExitApp();

break;

case CommandType.CreateFrame:

CreateFrame(userInterface, frameNames);

break;

case CommandType.CreateSlot:

Slot slot = CreateSlot(userInterface);

TryExecLisps(availableLisps, \_openedFrame, slot, userInterface);

break;

case CommandType.OpenFrame:

OpenFrame(userInterface, frameNames);

break;

case CommandType.PrintAllFrames:

userInterface.PrintAllFrames(\_createdFrames);

break;

case CommandType.SaveToDisk:

frameSerializer.Save(\_createdFrames);

userInterface.PrintSuccessDiskSave();

break;

case CommandType.FindFrameByName:

frameFinder.FindByFrameName(\_createdFrames);

break;

case CommandType.FindFrameBySlotName:

frameFinder.FindBySlotName(\_createdFrames);

break;

case CommandType.FindFrameBySlotValue:

frameFinder.FindBySlotValue(\_createdFrames);

break;

case CommandType.RedactSlotValue:

userInterface.RequestSlotEditing(\_openedFrame);

break;

case CommandType.CloseFrame:

CloseFrame(userInterface);

break;

case CommandType.DeleteFrame:

DeleteFrame(userInterface);

break;

case CommandType.SetInheritance:

SetInheritance(userInterface, frameNames, inheritTypeLisps);

break;

default: Console.WriteLine($"Command {command} not implemented");

break;

}

}

}

private static void SetInheritance(UserInterface userInterface, IEnumerable<string> frameNames,

IEnumerable<Lisp> availableLisps)

{

Frame parentFrame = GetFrameByInputName(userInterface, frameNames, "родителя");

if (parentFrame.Name == \_openedFrame.Name)

{

Console.WriteLine("Фрейм не может наследоваться от себя! Попробуйте ещё раз.");

SetInheritance(userInterface, frameNames, availableLisps);

return;

}

\_openedFrame.SetInheritance(parentFrame);

userInterface.PrintSuccessInhertaince(\_openedFrame.Name, parentFrame.Name);

TryExecLisps(availableLisps, \_openedFrame, null, userInterface);

}

private static void DeleteFrame(UserInterface userInterface)

{

string frameName = \_openedFrame.Name;

\_createdFrames.Remove(\_openedFrame);

\_openedFrame = null;

userInterface.PrintSuccessDelete(frameName);

}

private static void OpenFrame(UserInterface userInterface, IEnumerable<string> createdFrameNames)

{

Frame foundFrame = GetFrameByInputName(userInterface, createdFrameNames);

\_openedFrame = foundFrame;

}

private static Frame GetFrameByInputName(UserInterface userInterface, IEnumerable<string> createdFrameNames,

string additionalMessage = "")

{

string frameName = userInterface.RequestCreatedFrameName(createdFrameNames, additionalMessage);

Frame foundFrame = \_createdFrames.First(frame => frame.Name == frameName);

return foundFrame;

}

private static Slot CreateSlot(UserInterface userInterface)

{

Slot slot = userInterface.RequestSlotCreation(\_openedFrame);

\_openedFrame.AddSlot(slot);

return slot;

}

private static void ExitApp()

{

Environment.Exit(0);

}

private static void CreateFrame(UserInterface userInterface, IEnumerable<string> frameNames)

{

string frameName = userInterface.RequestFrameCreationName(frameNames);

\_createdFrames.Add(new Frame(frameName));

userInterface.PrintSuccessFrameCreation(frameName);

}

private static void CloseFrame(UserInterface userInterface)

{

string frameName = \_openedFrame.Name;

\_openedFrame = null;

userInterface.PrintFrameClosed(frameName);

}

private static void TryExecLisps(IEnumerable<Lisp> availableLisps, Frame slotAddedTo, Slot addedSlot,

UserInterface userInterface)

{

var executableLisps = availableLisps

.Where(lisp => lisp.CanBeExecuted(slotAddedTo, addedSlot));

foreach (Lisp lisp in executableLisps)

{

userInterface.PrintExecutedLisp(lisp);

}

}

}

}

***Файл FrameSerializer.cs***

using FrameKnowledgeModel.Data;

using Newtonsoft.Json;

using Newtonsoft.Json.Linq;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

namespace FrameKnowledgeModel

{

public class FrameSerializer

{

private readonly string \_filePath;

public FrameSerializer(string filePath)

{

\_filePath = filePath;

}

public void Save(IReadOnlyList<Frame> frames)

{

using StreamWriter streamWriter = new StreamWriter(new FileStream(\_filePath, FileMode.Truncate, FileAccess.Write));

string jsonFrames = JsonConvert.SerializeObject(frames);

streamWriter.Write(jsonFrames);

}

public List<Frame> Load()

{

using StreamReader streamReader = new StreamReader(new FileStream(\_filePath, FileMode.Open, FileAccess.Read));

string fileContent = streamReader.ReadToEnd();

List<Frame> fileFrames = new List<Frame>();

fileFrames = JsonConvert.DeserializeObject<List<Frame>>(fileContent);

IEnumerable<Slot> lispSlots = fileFrames

.SelectMany(frame => frame.Slots)

.Where(slot => slot.DataType == DataType.Lisp);

foreach (var slot in lispSlots)

{

JObject data = slot.DataValue as JObject;

string json = data.ToString();

slot.DataValue = JsonConvert.DeserializeObject<Lisp>(json);

}

return fileFrames;

}

}

}

***Файл FrameFinder.cs***

using FrameKnowledgeModel.Data;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace FrameKnowledgeModel

{

public class FrameFinder

{

private readonly UserInterface \_userInterface;

public FrameFinder(UserInterface userInterface)

{

\_userInterface = userInterface;

}

public void SetStartingFrameInheritance(IEnumerable<Frame> frames)

{

foreach (var frame in frames)

{

if (frame.ParentName == null) continue;

Frame parentFrame = frames.FirstOrDefault(f => f.Name == frame.ParentName);

frame.SetInheritance(parentFrame);

}

}

public void FindByFrameName(IEnumerable<Frame> frames)

{

\_userInterface.RequestIsFrameCreatedCheck(frames.Select(frame => frame.Name));

}

public void FindBySlotValue(IEnumerable<Frame> frames)

{

string slotValue = \_userInterface.RequestAnySlotValue();

IEnumerable<IReadOnlySlot> foundSlots = frames.SelectMany(frame => frame.Slots)

.Where(slot => slot.DataValue.ToString() == slotValue);

IEnumerable<Frame> foundFrames = frames.Where(frame => frame.Slots.Intersect(foundSlots).Any());

\_userInterface.PrintFoundFrames(foundFrames);

}

public void FindBySlotName(IEnumerable<Frame> frames)

{

var framesWithSlot = GetFramesWithSlotName(frames, out \_, out \_);

\_userInterface.PrintFoundFrames(framesWithSlot);

}

private IEnumerable<Frame> GetFramesWithSlotName(IEnumerable<Frame> frames, out IReadOnlySlot commonSlot,

out string slotName)

{

string slotNamee = slotName = \_userInterface.RequestSlotName();

var framesWithSlot = frames.Where(frame => frame.Slots.Any(slot => slot.Name == slotNamee));

commonSlot = framesWithSlot.First().Slots.First(slot => slot.Name == slotNamee);

return framesWithSlot;

}

}

}

***Файл Extensions.cs***

using System;

using System.Collections.Generic;

using System.Linq;

namespace FrameKnowledgeModel

{

public static class Extenstions

{

public static T RequestEnumValue<T>(string printMessage, string warningMessage,

IEnumerable<T> toExclude = null) where T : Enum

{

Type enumType = typeof(T);

T[] enumFitsValues = (T[])Enum.GetValues(typeof(T));

if (toExclude != null && toExclude.Any())

{

enumFitsValues = enumFitsValues.Where(type => !toExclude.Contains(type)).ToArray();

}

IEnumerable<string> enumStrings = enumFitsValues.Select(enumValue => enumValue.ToString());

string enumTypesStr = $"[{string.Join(", ", enumStrings)}]";

Console.WriteLine($"{printMessage} {enumTypesStr}: ");

string enumTypeReply = Console.ReadLine();

while (!enumStrings.Contains(enumTypeReply))

{

Console.WriteLine(warningMessage);

enumTypeReply = Console.ReadLine();

}

return (T)Enum.Parse(enumType, enumTypeReply);

}

public static string ToSingleStr(this IEnumerable<string> stringCollection, string separator = ", ") =>

string.Join(separator, stringCollection);

}

}

***Файл Frame.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace FrameKnowledgeModel.Data

{

[Serializable]

public class Frame

{

private readonly List<Slot> \_slots = new List<Slot>();

private readonly Dictionary<string, object> \_uniqueSlotValues = new Dictionary<string, object>();

private Frame \_parent;

public Frame(string name)

{

Name = name;

}

public string Name { get; set; }

public string ParentName => \_parent?.Name;

public IEnumerable<Slot> Slots

{

get => \_parent is null ? \_slots : \_slots.Concat(\_parent.Slots

.Select(slot => slot.InheritType == InheritType.Unique

? CopyUniqueSlot(slot)

: slot));

}

public void AddSlot(Slot slot) => \_slots.Add(slot);

public bool HasSlot(IReadOnlySlot slot) => Slots.Any(sl => sl.Name == slot.Name);

public void EditSlotValue(IReadOnlySlot slotToEdit, object slotValue)

{

if (!HasSlot(slotToEdit))

{

throw new ArgumentException($"Фрейм {Name} не имеет слота {slotToEdit.Name}, запрашимаего " +

$"для редактирования");

}

Slot slot = slotToEdit as Slot;

if (\_parent?.HasSlot(slot) ?? false)

{

if (slot.InheritType == InheritType.Same)

{

\_parent.EditSlotValue(slotToEdit, slotValue);

}

else

{

\_uniqueSlotValues[slot.Name] = slotValue;

}

}

else

{

slot.DataValue = slotValue;

}

}

public void SetInheritance(Frame parent) => \_parent = parent;

private Slot CopyUniqueSlot(Slot slot)

{

if (!\_uniqueSlotValues.TryGetValue(slot.Name, out object value))

{

value = slot.DataValue;

}

return new Slot(slot.Name, slot.InheritType, slot.DataType, value);

}

}

}

***Файл Slot.cs***

using System;

namespace FrameKnowledgeModel.Data

{

public enum InheritType

{

Unique, Same

}

public enum DataType

{

Real, Integer, Boolean, Text, Frame, Lisp

}

[Serializable]

public class Slot : IReadOnlySlot

{

public Slot(string name, InheritType inheritType, DataType dataType, object dataValue)

{

Name = name;

InheritType = inheritType;

DataType = dataType;

DataValue = dataValue;

}

public string Name { get; }

public InheritType InheritType { get; }

public DataType DataType { get; }

public object DataValue { get; set; }

}

public interface IReadOnlySlot

{

public string Name { get; }

public InheritType InheritType { get; }

public DataType DataType { get; }

public object DataValue { get; }

}

}

***Файл Lisp.cs***

using System;

namespace FrameKnowledgeModel.Data

{

public enum LispType

{

ThisFrame, SlotName, SlotValue, Inherit

}

[Serializable]

public class Lisp

{

public Lisp(string ownerName, string printMessage, LispType lispType,

string execSlotName = null, object execSlotValue = null, string execInheritorName = null)

{

OwnerName = ownerName;

PrintMessage = printMessage;

LispType = lispType;

ExecSlotName = execSlotName;

ExecSlotValue = execSlotValue;

ExecInheritorName = execInheritorName;

}

public string PrintMessage { get; }

public string OwnerName { get; }

public LispType LispType { get; }

public string ExecSlotName { get; }

public object ExecSlotValue { get; }

public string ExecInheritorName { get; }

public bool CanBeExecuted(Frame slotAddedTo, Slot addedSlot)

{

switch (LispType)

{

case LispType.ThisFrame:

return slotAddedTo.Name == OwnerName;

case LispType.SlotName:

return addedSlot.Name == ExecSlotName;

case LispType.SlotValue:

return addedSlot.DataValue == ExecSlotValue;

case LispType.Inherit:

return slotAddedTo.ParentName == ExecInheritorName;

default:

throw new NotImplementedException();

}

}

public override string ToString()

{

return PrintMessage;

}

}

}