МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
"МИРЭА - Российский технологический университет"
РТУ МИРЭА

Институт Информационных Технологий
Кафедра Вычислительной Техники

ПРАКТИЧЕСКАЯ РАБОТА №1
«Онтология»

по дисциплине
«Системный анализ данных СППР»

Студент группы: ИКБО-04-22 Основин А.И.
(Ф. И.О.студента)

Преподаватель __Железняк Л.М.__
(Ф.И.О. преподавателя)

Москва 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Возникновение онтологий и их стремительное развитие связано с проявлением в нашей реальности следующих новых факторов:

- колоссальный рост объемов информации, предъявляемых для обработки (анализа, использования) специалистам самых различных областей деятельности;
- чрезвычайная зашумленность этих потоков (повторы, противоречивость, разноуровневость, и т.п.);
- острая необходимость в использовании одних и тех же знаний разными специалистами в разных целях;
- всеобщая интернетизация нашей жизни и острая необходимость в структуризации информации для её представления пользователям и более эффективного поиска;
- необходимость сокращения времени на поиск нужной информации и повышения качества информационных услуг в Интернете.

Онтологии – это базы знаний специального типа, которые могут читаться и пониматься, отчуждаться от разработчика и/или физически разделяться их пользователями.

Существует много видов онтологий, однако одним из самых широко применяемых видов являются онтологии предметных областей, содержащие понятия определённой области знаний или входящих в неё областей.

1 ОНТОЛОГИЯ

Построение баз знаний является основой формализации экспертных систем, и одним из наиболее четких примеров баз знаний является онтология. Онтология направлена на стандартизацию и унификацию представления знаний. В онтологии описываются основные элементы предметной области, организованные в виде тройки: $O = \langle X, R, F \rangle$, где:

X — конечное множество концептов (понятий, терминов) предметной области;

R — конечное множество отношений между этими концептами;

F — конечное множество функций интерпретации (аксиом), которые определяют взаимосвязи и правила, действующие на концептах и отношениях.

В формализованном виде онтологии представляются через:

Классы — абстракции, описывающие категории объектов или понятий;

Слоты (атрибуты) — характеристики или свойства, которыми обладают классы;

Экземпляры — конкретные объекты, являющиеся представителями классов.

Такое формальное представление онтологий позволяет моделировать знания в системах искусственного интеллекта и обеспечивать их совместное использование и обработку.

1.1 Цель практической работы

Целью данной практической работы является ознакомление с построением некоторой базы знаний на основе подобия фреймовой системы, а также ее реализация, анализ и применение на практике.

1.2 Постановка задачи

Необходимо разработать онтологию выбранной предметной области – «Онлайн магазин одежды». Данная предметная область была выбрана из-за личного интереса к теме.

1.3 Описание онтологии

Рассматриваемый онлайн магазин создан для продажи мужской одежды стиля кэжуал (офисной, официальной). Такой одеждой являются штаны, рубашки и туфли. Ассортимент одежды предоставляют магазины (продавцы), зарегистрированные на площадке. У каждого продавца собственный ассортимент товаров. Пользователи площадки могут выбрать товары, исходя из ассортимента всего онлайн магазина. У каждого пользователя есть список приобретённых товаров.

На основе этого описания можно составить онтологию, состоящую из следующих классов:

- «Онлайн магазин одежды» — общий базовый класс для всех классов;
- «Продукт» — базовый класс для разных видов товаров, представленных в онлайн магазине, содержит общие слоты «Бренд», «Модель», «Цена», «Цвет», «Размер»;
- «Штаны» — класс для описания штанов, содержит слоты «Тип посадки» и «Хорошо сидит с», ссылающийся на экземпляр класса «Туфли»;
- «Рубашка» — класс для описания рубашек, содержит слоты «Количество пуговиц» и «Стильно смотрится с», ссылающийся на экземпляр класса «Штаны»;
- «Туфли» — класс для описания туфель, содержит слот «Высота каблука»;

- «Пользователь» — базовый класс для всех видов пользователей, содержит общие слоты «Имя» и «Активен» (для продавца – продаёт ли он сейчас товары; для покупателя – покупает ли он товары);
- «Продавец» — класс для описания продавца (магазина с собственным ассортиментом), содержит слот «Список товаров», ссылающийся на несколько экземпляров класса «Продукт»;
- «Покупатель» — класс для описания покупателя, содержит слот «Выкуп» и «Список покупок», ссылающийся на несколько экземпляров класса «Продукт».

Данное описание использовано для построения графической схемы онтологии (Рисунок 1.3.1).

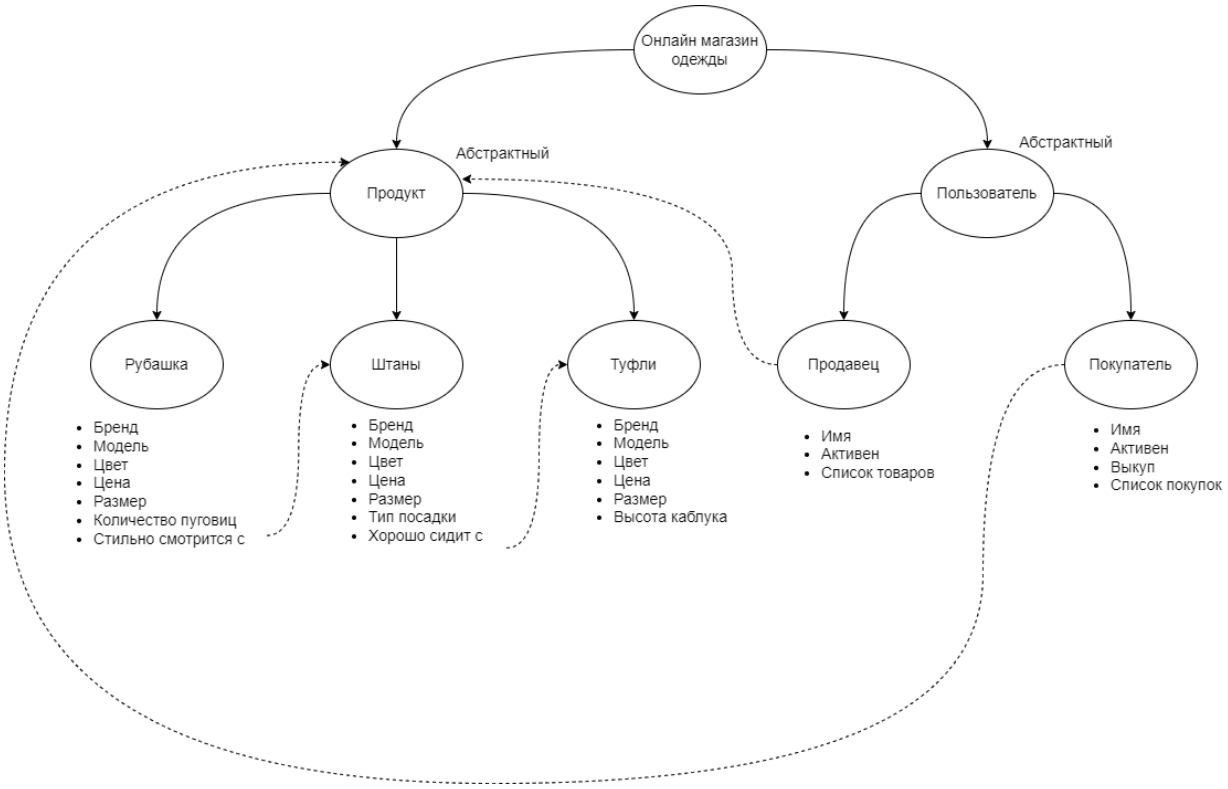

Рисунок 1.3.1 – Схема онтологии «Онлайн магазин одежды»

1.4 Построение онтологии в Protégé

Для подробного изучения составленной онтологии использован инструмент для построения, редактирования онтологий и работы с ними

Protégé. Сначала созданы классы (Рисунок 1.4.1), а затем в них описаны слоты (Рисунок 1.4.2–1.4.6).

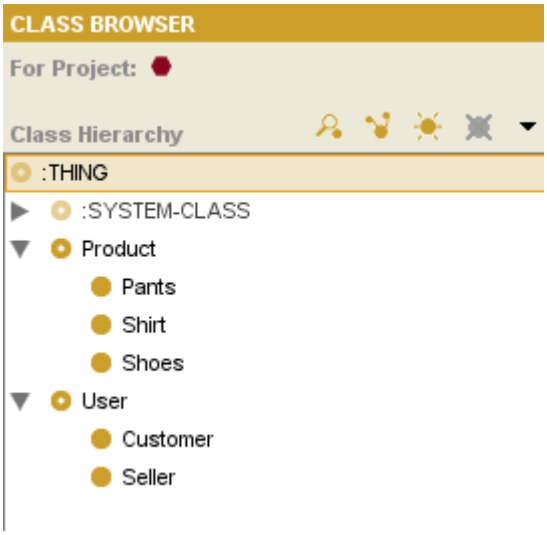

Рисунок 1.4.1 – Составленная иерархия классов

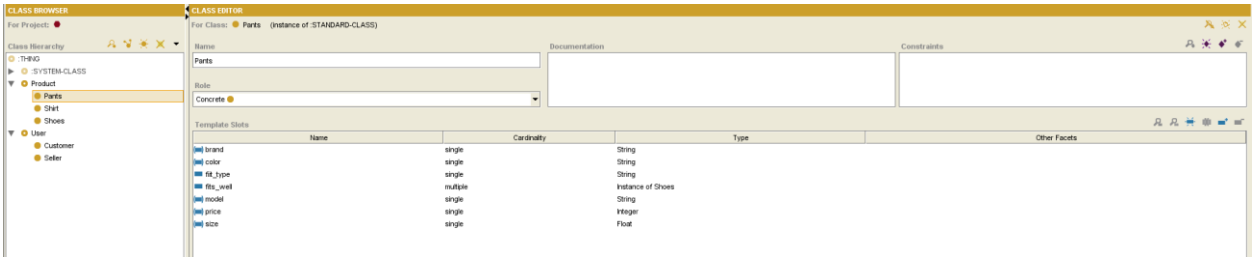

Рисунок 1.4.2 – Слоты класса «Штаны»

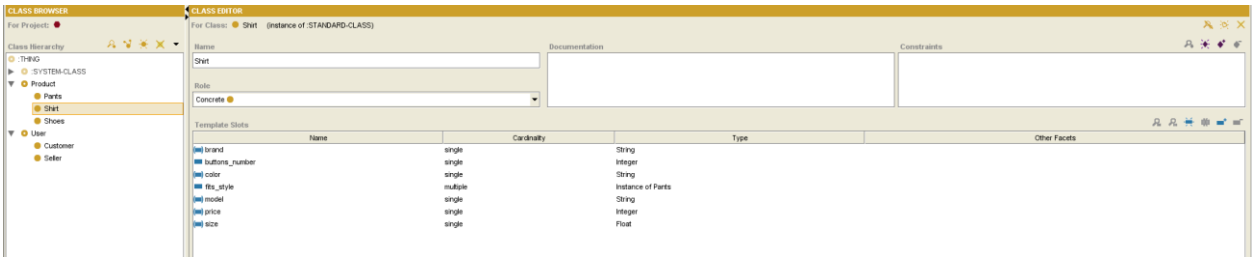

Рисунок 1.4.3 – Слоты класса «Рубашка»

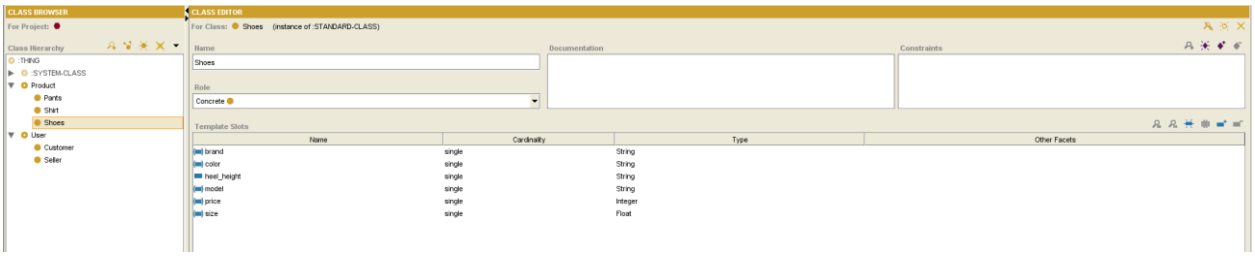

Рисунок 1.4.4 – Слоты класса «Туфли»

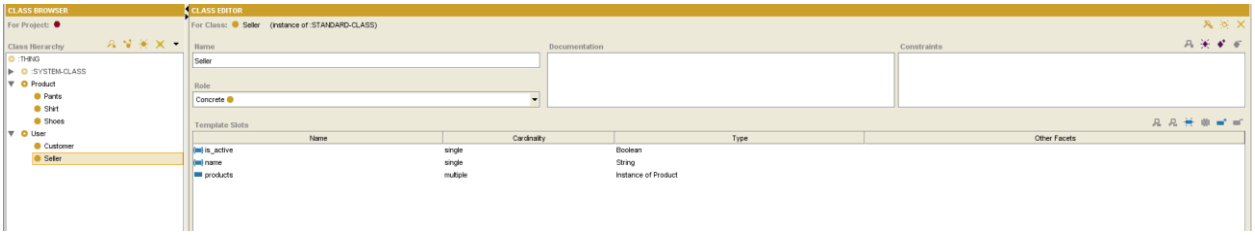

Рисунок 1.4.5 – Слоты класса «Продавец»

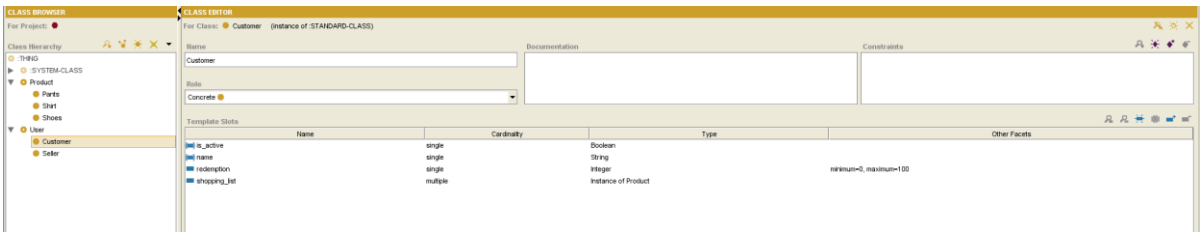

Рисунок 1.4.6 – Слоты класса «Покупатель»

После составления и описания классов созданы экземпляры каждого из классов (Рисунок 1.4.7-1.4.11).

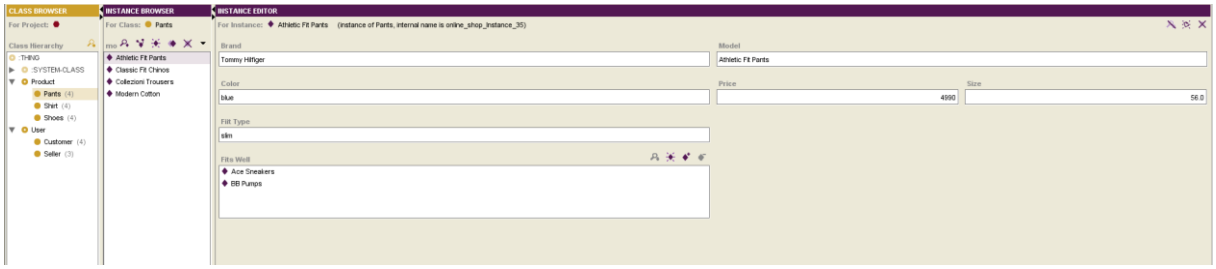

Рисунок 1.4.7 – Экземпляры класса «Штаны»

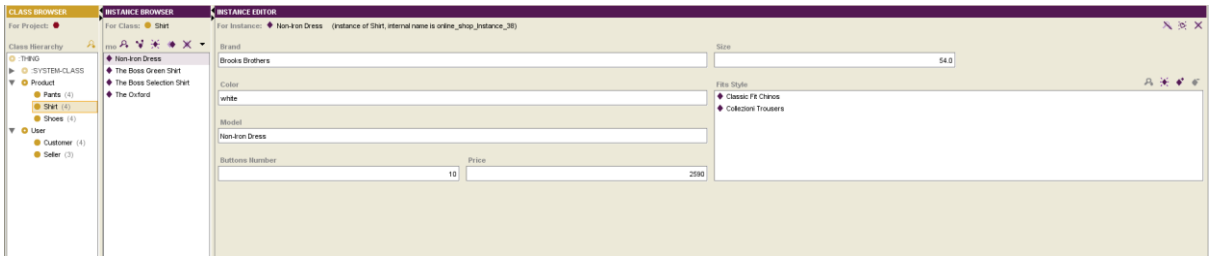

Рисунок 1.4.8 – Экземпляры класса «Рубашка»

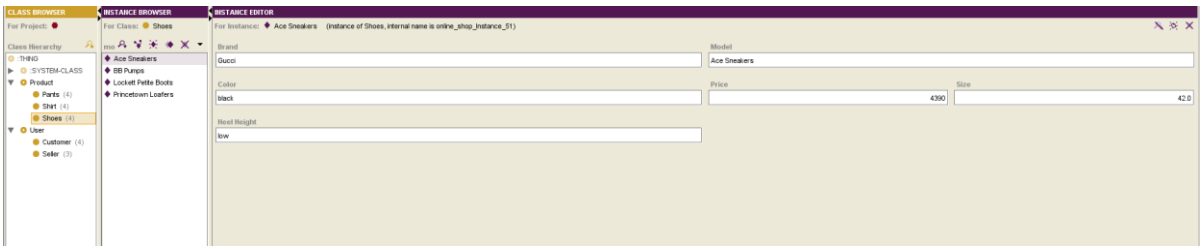

Рисунок 1.4.9 – Экземпляры класса «Туфли»

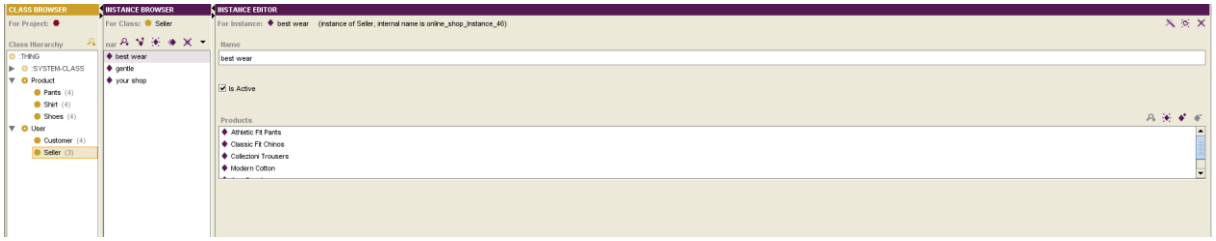

Рисунок 1.4.10 – Экземпляры класса «Продавец»

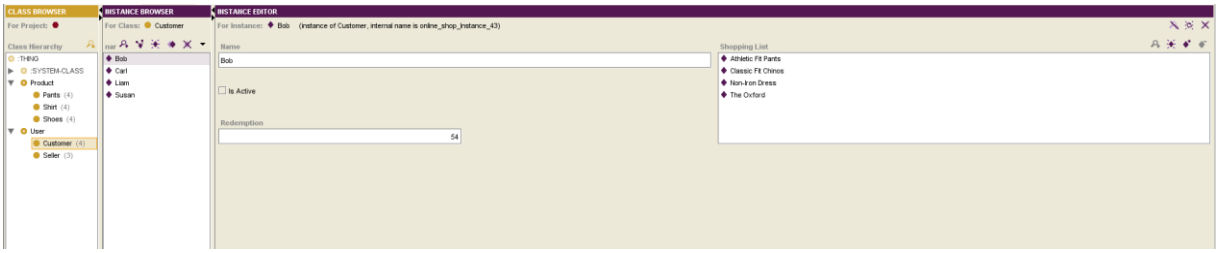

Рисунок 1.4.11 – Экземпляры класса «Покупатель»

1.5 Выполнение запросов в Protege

Программа Protégé позволяет составлять запросы на получение объектов по определённым условиям, а также вытаскивать связанные объекты для уже полученных объектов. Проделаны обычные запросы на получение экземпляров (Рисунок 1.5.1), а также сделаны цепные запросы на получение связанных объектов (Рисунок 1.5.2–1.5.3).

Рисунок 1.5.1 – Одинарный запрос на получение экземпляров класса «Штаны»

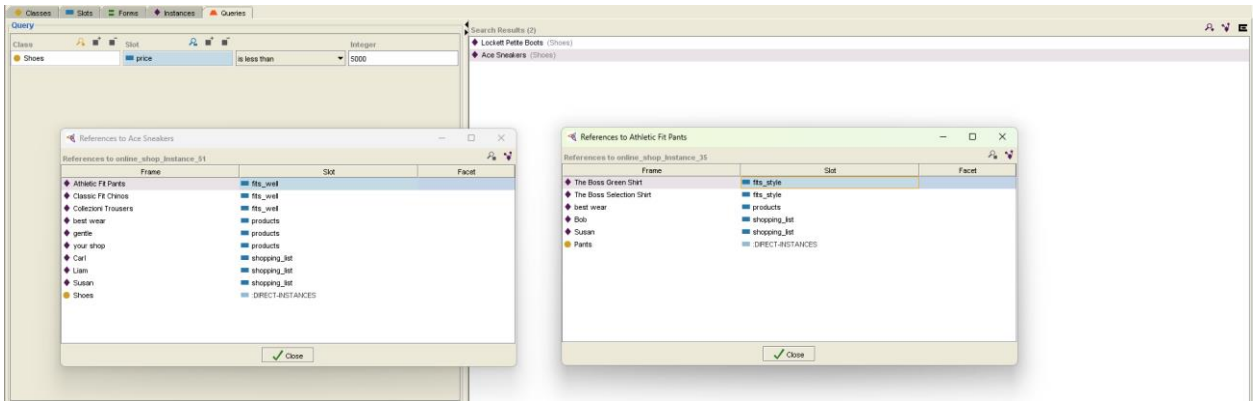

Рисунок 1.5.2 – Цепной запрос на получение рубашек, смотрящихся стильно со штанами, которые хорошо сидят с туфлями

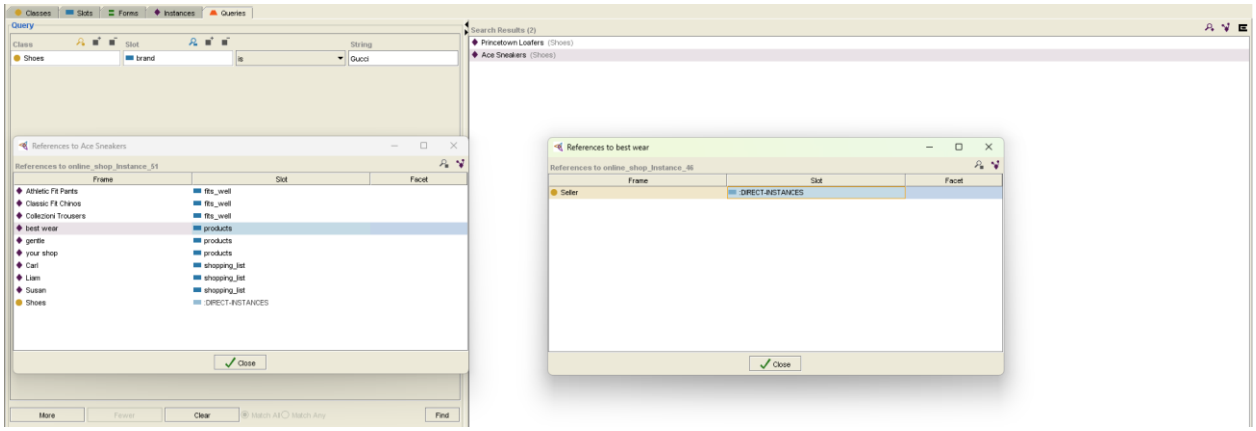

Рисунок 1.5.3 – Цепной запрос на получение магазина, продающего туфли

1.6 Результаты выполнения программного кода

Для работы с онтологиями написана программа на языке программирования Python, которая запускается в консоли и поддерживает выполнение запросов на получение экземпляров. Работы программы продемонстрирована на Рисунках 1.6.1–1.6.2.

Рисунок 1.6.1 – Результат выполнения программы

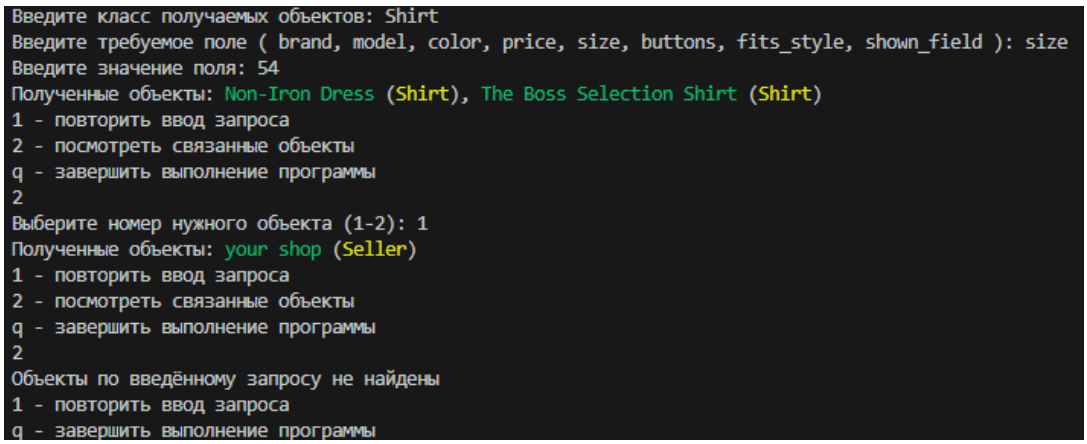

Рисунок 1.6.2 – Результат выполнения программы

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной практической работы изучены теоретические основы системного анализа и использования онтологий в широком ряде задач, получены навыки построения онтологий и работы с ними, включая создание классов для описания выбранной предметной области, создание слотов в классах и создание экземпляров. С помощью инструменты работы с онтологиями Protégé выполнены запросы на получение объектов по различным запросам.

В качестве закрепления полученных знаний написана программа на языке программирования Python, способная работать с онтологией выбранной предметной области. В её функционал входит возможность писать запросы на получение экземпляров и связанных объектов.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Сорокин А.Б. Введение в роевой интеллект: теория, расчеты и приложения [Электронный ресурс]: Учебно-методическое пособие — М.: Московский технологический университет (МИРЭА), 2019. — 1 электрон. опт. диск (CD-ROM)
2. Лапшин В. А. Онтологии в компьютерных системах. — М.: Научный мир, 2010.
3. Карпенко А. П. Современные алгоритмы поисковой оптимизации. Алгоритмы, вдохновленные природой: учебное пособие / А. П. Карпенко. – Москва: Издательство МГТУ им. Н. Э. Баумана, 2014. – 448 с.
4. Noy, Natalya F.; McGuinness, Deborah L. (March 2001). "Ontology Development 101: A Guide to Creating Your First Ontology". Stanford Knowledge Systems Laboratory Technical Report KSL-01-05, Stanford Medical Informatics Technical Report SMI-2001-0880
5. Добров Б. В., Иванов В.В., Лукашевич Н.В., Соловьев В.Д. Онтологии и тезаурусы: модели, инструменты, приложения. — М.: Бином. Лаборатория знаний, 2009. — 173 с. — ISBN 978-5-9963-0007-5.

ПРИЛОЖЕНИЯ

Приложение А — Код реализации онтологии на языке Python

Приложение А

Код реализации онтологии на языке Python

Листинг А.1 – Реализация онтологии

```
from typing import Any
import random
import re

class OntologyObject():
    shown_field = 'name'

    def __str__(self) -> str:
        if isinstance(self, Product):
            self.shown_field = 'model'
        return getattr(self, self.shown_field)

class Product(OntologyObject):
    instances = []

    def __init__(self, brand: str, model: str, color: str, price: int, size:
float) -> None:
        self.brand = brand
        self.model = model
        self.color = color
        self.price = price
        self.size = size
        Product.instances.append(self)

class User(OntologyObject):
    instances = []

    def __init__(self, name: str, is_active: bool):
        self.name = name
        self.is_active = is_active
        User.instances.append(self)

class Shoes(Product):
    instances = []

    def __init__(self, brand: str, model: str, color: str, price: int, size:
float, heel_height: str):
        super().__init__(brand, model, color, price, size)
        self.heel_height = heel_height
        Shoes.instances.append(self)

class Pants(Product):
    instances = []

    def __init__(self, brand: str, model: str, color: str, price: int, size:
float, fit_type: str, fits_well: list[Shoes]):
        super().__init__(brand, model, color, price, size)
        self.fit_type = fit_type
        self.fits_well = fits_well
        Pants.instances.append(self)

class Shirt(Product):
    instances = []
```

Продолжение Листинга А.1

```
        def __init__(self, brand: str, model: str, color: str, price: int,
size: float, buttons: int, fits_style: list[Pants]):
            super().__init__(brand, model, color, price, size)
            self.buttons = buttons
            self.fits_style = fits_style
            Shirt.instances.append(self)

    class Seller(User):
        instances = []

        def __init__(self, name: str, is_active: bool, products:
list[Product]):
            super().__init__(name, is_active)
            self.products = products
            Seller.instances.append(self)

    class Customer(User):
        instances = []

        def __init__(self, name: str, is_active: bool, redemption: int,
shopping_list: list[Product]):
            super().__init__(name, is_active)
            self.redemption = redemption
            self.shopping_list = shopping_list
            Customer.instances.append(self)

    def find_related_objects_by_value(cls: OntologyObject.__class__,
                                      lookup_field: str, value: Any):
        instances = cls.instances
        result = []
        for instance in instances:
            if isinstance(getattr(instance, lookup_field), OntologyObject):
                if not isinstance(value, OntologyObject):
                    value = find_object_by_name(
                        getattr(instance, lookup_field).__class__, value)
                    if value is None:
                        return None

            if isinstance(getattr(instance, lookup_field), bool):
                value = bool(value)
            elif isinstance(getattr(instance, lookup_field), int):
                value = int(value)
            elif isinstance(getattr(instance, lookup_field), float):
                value = float(value)

            if isinstance(getattr(instance, lookup_field), list):
                for subfield in getattr(instance, lookup_field):
                    if subfield == value:
                        result.append((instance, type(instance).__name__))
            else:
                if getattr(instance, lookup_field) == value:
                    result.append((instance, type(instance).__name__))
        return result

    def find_object_by_name(cls: OntologyObject.__class__, name: str):
        instances = cls.instances
        for instance in instances:
            if instance.name == name:
                return instance

        return None
```

Продолжение Листинга А.1

```
    def get_class(class_name: str):
        classes = {
            'shirt': Shirt,
            'pants': Pants,
            'shoes': Shoes,
            'seller': Seller,
            'customer': Customer
        }

        class_name = class_name.lower().strip()
        if class_name in classes:
            return classes[class_name]
        else:
            return None

    def get_random_class_instance(cls: OntologyObject.__class__):
        instances = cls.instances
        return random.choice(instances)

    def get_related_class(obj: OntologyObject):
        classes = [Shirt, Pants, Shoes, Seller, Customer]
        class_types = set()
        for cls in classes:
            cls_instances = cls.instances
            for instance in cls_instances:
                fields = [
                    key for key in instance.__dict__.keys()
                    if not re.match(r"__\w*__", key)
                ]
                for field in fields:
                    field_value = getattr(instance, field)
                    if field_value == obj:
                        class_types.add((cls, field))
                    elif isinstance(field_value, list):
                        for subfield_value in field_value:
                            if subfield_value == obj:
                                class_types.add((cls, field))

        return list(class_types)

    def main():
        shoes = [
            Shoes('Gucci', 'Ace Sneakers', 'black', 4390, 42., 'low'),
            Shoes('Manolo Blahnik', 'BB Pumps', 'black', 5290, 40., 'flat'),
            Shoes('Jimmy Choo', 'Lockett Petite Boots', 'pink', 4890, 42.,
'high'),
            Shoes('Gucci', 'Princetown Loafers', 'brown', 5290, 44., 'flat')
            ]
        pants = [
            Pants('Tommy Hilfiger', 'Athletic Fit Pants', 'blue', 4990, 56.,
'slim', [shoes[0], shoes[1]]),
            Pants('Tommy Hilfiger', 'Classic Fit Chinos', 'black', 3890,
58., 'regular', [shoes[0], shoes[3]]),
            Pants('Armani', 'Collezioni Trousers', 'black', 4590, 50.,
'regular', [shoes[0], shoes[1], shoes[2], shoes[3]]),
            Pants('Calvin Klein', 'Modern Cotton', 'brown', 2990, 52.,
'slim', [shoes[3]])
            ]
        shirts = [
            Shirt('Brooks Brothers', 'Non-Iron Dress', 'white', 2590, 54.,
10, [pants[1], pants[2]]),
```

Продолжение Листинга А.1

```
            Shirt('Hugo Boss', 'The Boss Green Shirt', 'white', 2690, 56.,
10, [pants[0], pants[1], pants[2], pants[3]]),
            Shirt('Hugo Boss', 'The Boss Selection Shirt', 'black', 3290,
54., 12, [pants[0], pants[3]]),
            Shirt('Thomas Pink', 'The Oxford', 'pink', 2290, 58., 12,
[pants[3]])
            ]
        sellers = [
            Seller('best wear', True, [pants[0], pants[1], pants[2],
pants[3], shoes[0], shoes[3]]),
            Seller('gentle', True, [shoes[0], shoes[2], shoes[3]]),
            Seller('your shop', False, [shirts[0], shirts[1], shirts[2],
shoes[1]]),
            ]
        customers = [
            Customer('Bob', False, 54, [pants[0], pants[1]]),
            Customer('Carl', True, 79, [shoes[0], shoes[3]]),
            Customer('Liam', True, 98, [shoes[0], shirts[2], pants[1]]),
            Customer('Susan', True, 97, [pants[0], pants[2], shirts[2]]),
            ]

        while True:
            while True:
                class_name = input('Введите класс получаемых объектов: ')
                cls: OntologyObject.__class__ | None = get_class(class_name)
                if cls is None:
                    print('Такого класса не существует\n\n')
                else:
                    break

            while True:
                instance = get_random_class_instance(cls)
                available_fields = [
                    key for key in instance.__dict__.keys()
                    if not re.match(r"__\w*__", key)
                ]
                field = input(
                    f'Введите требуемое поле ( {", ".join(available_fields)}
): ')
                try:
                    getattr(instance, field)
                    break
                except Exception:
                    print('Такого поля в классе не существует\n\n')

            value = input('Введите значение поля: ')
            res = find_related_objects_by_value(cls, field, value)

            while True:
                flag = False
                if res is None or len(res) == 0:
                    print('Объекты по введённому запросу не найдены')
                    while True:
                        _type = input(
                            '1 - повторить ввод запроса\nq - завершить
выполнение программы\n'
                        )
                        if _type == '1':
                            break
                        elif _type == 'q':
                            exit(0)
```

Продолжение Листинга А.1

```
                else:
                    str_objects = [
                        "\033[32m" + str(obj) + "\033[0m (\033[33m" +
                        str(obj_type) + "\033[0m)" for obj, obj_type in res
                    ]
                    print(f'Полученные объекты: {", ".join(str_objects)}')
                    while True:
                        _type = input('1 - повторить ввод запроса\n'
                                      '2 - посмотреть связанные объекты\n'
                                      'q - завершить выполнение
программы\n')
                        if _type == '1':
                            flag = False
                            break
                        elif _type == '2':
                            flag = True
                            break
                        elif _type == 'q':
                            exit(0)

                    if flag:
                        if len(res) == 1:
                            obj_number = 1
                        else:
                            while True:
                                obj_number = int(
                                    input(
                                        f'Выберите номер нужного объекта (1-
{len(res)}): '
                                    ))
                                if obj_number not in range(1, len(res) + 1):
                                    print('Введён неправильный номер')
                                else:
                                    break

                        instance = res[obj_number - 1][0]
                        related_classes = get_related_class(instance)
                        res = []
                        for rel_class, field_name in related_classes:
                            res.extend(
                                find_related_objects_by_value(
                                    rel_class, field_name, instance))
                    else:
                        break
                if not flag:
                    break

    if __name__ == "__main__":
        main()
```