Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Информационных технологий и управления
Кафедра Интеллектуальных информационных технологий

ОТЧЁТ

по лабораторной работе №2
по дисциплине “Проектирование программного обеспечения в интеллектуальных системах”

Выполнили:
Р. В. Липский, гр. 121701
Проверил:
С. В. Бутрин

Цель и постановка задачи

Цель: Изучить построение графического пользовательского интерфейса

Задание:
Разработать оконное приложение с одним главным окном и несколькими дочерними диалогами. Вызов диалогов осуществляется через соответствующие пункты меню. Команды меню должны дублироваться на панели инструментов.

Общее требования к выполнению:

1. Приложение должно быть построено при помощи шаблона проектирования Model-View-Controller
2. Приложение должно уметь:
    2.1. Формировать массив записей путем ввода записей через окна диалога
    2.2. Осуществлять поиск записей в массиве в соответствие с условиями, указанными в отдельном диалоговом окне (не в диалоге удаления)
    2.3. В варианте задания условия поиска должны вводиться в специальном диалоговом окне, результат поиска выводиться в нём же с помощью стандартных элементов управления
    2.4. Удалять запись из массива по условиям, указанным в варианте. Условия удаления должны вводиться в отдельном диалоговом окне (не в диалоге поиска);
    2.5. Пользователю должно сообщаться о том были ли удалены записи, и сколько было удалено, согласно введенным условиям или таких записей не было найдено;
    2.6. Отображать весь текущий массив записей в главном окне приложения
    2.7. Сохранять и загружать массив записей из указываемого пользователем файла, посредством стандартного диалога сохранения/загрузки. Формат хранения данных в файле XML. Для записи использовать DOM парсер, для чтения SAX парсер.
    2.8. Для хранения должны использоваться правильные типы. Например, в таблице написано, что поле будет содержать дату рождения, значит, при реализации должен использоваться тип дата для данного поля. На диалогах для ввода даты должен использоваться специальный компонент для ввода дат. Например, календарь, чтобы дату было удобно создать. Если поле хранит число. Например, возраст, значит нужно использовать целочисленный атрибут для хранения возраста.
    2.9. Вывод записей в главном окне и в диалоге поиска осуществляется в постраничном виде. Например, по 10 записей на странице. Элемент управления страницами должен позволять переходить на первую, последнюю, следующую и предыдущую страницы, должен позволять изменить число записей на странице и показывать текущее число записей на странице, а также должен показывать число всех доступных записей и номер текущей активной страница, а также число всех доступных страниц.

2.10. При демонстрации работы должно быть несколько файлов, в каждом из которых сохранено минимум 50 уникальных записей. Каждая запись должна быть более-менее осмысленная. Варианты вида “ssdsds” не подходят.

Вариант 7

Значения атрибутов:
Состав: основной/запасной/ “n/a”.
Позиция: в зависимости от вида спорта.
Титулы: некоторое число.
Разряд: 1-й юношеский, 2-й разряд, 3й-разряд, кмс, мастер спорта.

ФИО спортсмена	Состав (если имеется)	Позиция	титулы	Вид спорта	Разряд

Условия поиска и удаления:

- по фио или виду спорта;
- по количеству завоеваний титула (при задании должны быть указаны верхний и нижний предел);
- по фио или разряду

Замечание: Список видов спорта и список видов разрядов в диалоге поиска, собирается системой и выводится в выпадающий список;

Реализация

RecordRepository

Абстрактный класс RecordRepository содержит методы для работы с записями. Класс является абстрактным, потому что он содержит абстрактные методы, которые не имеют реализации в этом классе, а должны быть определены в его наследниках.

Первая строка импортирует модуль ABC из библиотеки abc, который используется для создания абстрактных базовых классов.

Вторая строка импортирует класс Record из модуля model.record, который представляет собой запись в базе данных.

Третья строка импортирует класс RecordFilter из модуля repository.filter.record_filter, который используется для фильтрации записей.

Далее, определяется класс RecordRepository, который наследуется от абстрактного класса ABC. Класс содержит абстрактные методы:

- get_all - получение всех записей, возможна фильтрация по RecordFilter.
- get_page - получение страницы записей, где page - номер страницы, size - количество записей на странице, возможна фильтрация по RecordFilter.
- add_record - добавление новой записи в базу данных.
- remove_record - удаление записи по её имени.
- get_sports - получение множества видов спорта, которые есть в базе данных.
- get_ranks - получение множества разрядов, которые есть в базе данных.

Этот класс является базовым для всех репозиториев записей, которые будут использоваться в проекте. Как именно будут реализованы методы, зависит от конкретной реализации репозитория.

```python
from abc import ABC, abstractmethod

from model.record import Record
from repository.filter.record_filter import RecordFilter


class RecordRepository(ABC):

    @abstractmethod
    def get_all(self, filt: RecordFilter = None): pass

    @abstractmethod
    def get_page(self, page: int, size: int, fltr: RecordFilter = None): pass

    @abstractmethod
    def add_record(self, record: Record): pass

    @abstractmethod
    def remove_record(self, name: str): pass

    @abstractmethod
    def get_sports(self) -> set[str]: pass

    @abstractmethod
    def get_ranks(self) -> set[str]: pass
```

XmlRecordRepository

Класс XmlRecordRepository наследуется от класса RecordRepository и реализует его абстрактные методы. Класс представляет собой репозиторий записей, который хранит данные в формате XML.

Первые три строки импортируют необходимые модули: json, xml.dom.minidom и BeautifulSoup из bs4.

Далее, определяется класс XmlRecordRepository, который наследуется от класса RecordRepository. Конструктор класса принимает путь к файлу с данными, который передается в переменную path. Внутри конструктора создается пустой список self.storage, в который будут добавляться записи. Затем, данные из файла

считываются в переменную data, и на их основе создается объект BeautifulSoup, который используется для парсинга XML.

Далее, происходит итерация по всем записям в XML-файле, и для каждой записи создается объект Record, который добавляется в список self.storage.

- Метод __save используется для сохранения данных в XML-файл. Внутри метода создается объект minidom.Document(), который представляет собой пустой XML-документ. Затем, создается элемент <records>, который будет содержать все записи, и добавляется в документ. Затем, происходит итерация по всем записям в self.storage, и для каждой записи создается элемент <record>, который содержит атрибуты name, rank, sport, position, squad и titles. Затем, элемент <record> добавляется в элемент <records>. В конце, данные сохраняются в файл с помощью функции doc.toprettyxml(indent=" ").
- Метод get_all возвращает все записи, которые соответствуют фильтру RecordFilter. Если фильтр не задан, то возвращаются все записи.
- Метод get_page возвращает страницу записей, где page - номер страницы, size - количество записей на странице, и применяется фильтр RecordFilter. Если фильтр не задан, то возвращаются все записи.
- Метод add_record добавляет новую запись в репозиторий и сохраняет изменения в файле.
- Метод remove_record удаляет запись с указанным именем из репозитория и сохраняет изменения в файле.
- Методы get_ranks и get_sports возвращают множество уникальных значений поля rank и sport соответственно.

```
import json
from xml.dom import minidom

from bs4 import BeautifulSoup

from model.record import Record
from repository.filter.record_filter import RecordFilter
from repository.record_repository import RecordRepository


class XmlRecordRepository(RecordRepository):

    def __init__(self, path):
        self.storage = []
        self.path = path

        with open(self.path) as file:
            data = file.read()

        bs = BeautifulSoup(data, "xml")
        records = bs.find_all("record")

        for record in records:
            self.storage.append(Record(
```

```
                name=record["name"],
                rank=record["rank"],
                position=record["position"],
                sport=record["sport"],
                squad=record["squad"],
                titles=json.loads(record["titles"].replace("'", '"'))
            ))

    def __save(self):
        doc = minidom.Document()

        xml = doc.createElement("records")
        doc.appendChild(xml)

        for record in self.storage:
            entry = doc.createElement("record")
            entry.setAttribute("name", record.name)
            entry.setAttribute("rank", record.rank)
            entry.setAttribute("sport", record.sport)
            entry.setAttribute("position", record.position)
            entry.setAttribute("squad", record.squad)
            entry.setAttribute("titles", str(record.titles))
            xml.appendChild(entry)

        with open(self.path, 'w') as f:
            f.write(doc.toprettyxml(indent="    "))

    def get_all(self, filt: RecordFilter = None):
        if filt is not None:
            return filt.filter(self.storage)

    def get_page(self, page: int, size: int, fltr: RecordFilter = None):
        return self.get_all(fltr)[page * size:(page + 1) * size]

    def add_record(self, record: Record):
        self.storage.append(record)
        self.__save()

    def remove_record(self, name: str):
        self.storage = list(filter(lambda x: x.name != name, self.storage))
        self.__save()

    def get_ranks(self) -> set[str]:
        result = self.storage
        result = map(lambda x: x.rank, result)
        return set(result)

    def get_sports(self) -> set[str]:
        result = self.storage
        result = map(lambda x: x.sport, result)
        return set(result)
```

Record

Класс Record используется для хранения информации о спортивных записях. Класс создается с помощью декоратора @dataclass, который автоматически создает конструктор и методы __repr__, __eq__, __ne__, __lt__, __le__, __gt__, __ge__, используя аннотации типов для полей класса.

Класс Record имеет шесть полей:

- name - имя спортсмена или команды
- squad - название команды
- position - позиция спортсмена
- titles - список достижений спортсмена или команды
- sport - название вида спорта
- rank - рейтинг спортсмена или команды
- Все поля являются строковыми, за исключением поля titles, которое является списком строк.

Класс Record представляет собой простую структуру данных, которая используется для представления информации о спортивных записях в удобном формате.

```
from dataclasses import dataclass


@dataclass
class Record:
   name: str
   squad: str
   position: str
   titles: list[str]
   sport: str
   rank: str
```

RecordController

Класс RecordController является контроллером для управления записями в репозитории RecordRepository. Контроллер содержит методы для получения страницы записей, получения общего количества записей, добавления и удаления записей, а также получения списка рангов и видов спорта.

Контроллер принимает в конструкторе объект RecordRepository, который используется для доступа к данным. Метод get_page возвращает список записей на заданной странице с заданным размером и фильтром. Метод get_total_count возвращает общее количество записей, удовлетворяющих заданному фильтру. Метод add_record добавляет новую запись в репозиторий. Метод remove_record удаляет запись по имени. Методы get_ranks и get_sports возвращают список всех рангов и видов спорта, соответственно.

Все методы контроллера работают с объектами класса Record. Метод add_record принимает аргументы, необходимые для создания объекта Record, а метод get_page возвращает список записей в виде кортежей, сформированных с помощью функции record_to_tuple из модуля controller.converter.

```
from controller.converter import record_to_tuple
from model.record import Record
from repository.filter.record_filter import RecordFilter
from repository.record_repository import RecordRepository


class RecordController:

    def __init__(self, repository: RecordRepository):
        self.repository = repository

    def get_page(self, page: int, size: int, fltr: RecordFilter) ->
list[tuple]:
        records = self.repository.get_page(page, size, fltr)
        records = map(record_to_tuple, records)
        return list(records)

    def get_total_count(self, fltr: RecordFilter):
        records = self.repository.get_all(fltr)
        return len(records)

    def add_record(self, name, squad, position, titles: list, sport, rank):
        record = Record(name, squad, position, titles, sport, rank)
        self.repository.add_record(record)

    def remove_record(self, name):
        self.repository.remove_record(name)

    def get_ranks(self):
        return self.repository.get_ranks()

    def get_sports(self):
        return self.repository.get_sports()
```

MainView

Данный код представляет собой графический интерфейс пользователя (GUI) для управления записями о спортсменах. Он использует библиотеку Tkinter для создания виджетов и элементов управления GUI.

Класс MainView является основным компонентом GUI и содержит методы для создания и настройки элементов управления, а также обработчики событий для кнопок и других элементов.

Метод __make_table создает таблицу Treeview для отображения записей о спортсменах. Метод __make_paging_controls создает кнопки для навигации по страницам записей, а также метку для отображения общего количества записей, найденных в результате фильтрации. Метод __make_record_controls создает кнопки для добавления, удаления и удаления всех записей, отфильтрованных по заданным критериям. Метод __make_filter_controls создает элементы управления для фильтрации записей по имени, виду спорта, разряду и количеству титулов.

Метод __render_table обновляет содержимое таблицы Treeview на основе текущего состояния фильтра и текущей страницы.

Методы __handle_next_page и __handle_prev_page обрабатывают нажатия на кнопки навигации по страницам, изменяя значение переменной self.page и вызывая метод __render_table для обновления таблицы.

Метод __handle_add_record вызывает диалоговое окно AddRecordDialog для добавления новой записи, передавая метод __handle_add_record_submit в качестве обратного вызова для обработки события добавления записи.

Метод __handle_remove_record удаляет выделенные записи из таблицы Treeview и из хранилища данных.

Метод __handle_remove_all удаляет все записи, отфильтрованные по заданным критериям, из таблицы Treeview и из хранилища данных.

```
from tkinter import ttk
import tkinter as tk
from tkinter.messagebox import showinfo

from controller.record_controller import RecordController
from repository.filter.record_filter import RecordFilter
from view.add_record_dialog import AddRecordDialog

PAGE_SIZE = 10


class MainView:

    # Utils:

    def __clear_table(self):
        for child in self.table.get_children(""):
            self.table.delete(child)

    def __render_table(self, *args, **kwargs):
        self.__clear_table()

        sports = self.controller.get_sports()
        sports.add("")
        self.sport_filter["values"] = tuple(sports)
```

```
        ranks = self.controller.get_ranks()
        ranks.add("")
        self.rank_filter["values"] = tuple(ranks)

        titles = self.title_filter.get()
        titles = titles.split("-")
        titles = map(lambda x: x.strip(), titles)
        titles = list(titles)
        titles_filter = None
        if len(titles) == 2:
            try:
                min = int(titles[0])
                max = int(titles[1]) + 1
                titles_filter = (min, max)
            except ValueError:
                pass

        self.fltr = RecordFilter(
            name=self.name_filter.get(),
            sports=self.sport_filter.get(),
            rank=self.rank_filter.get(),
            titles_count=titles_filter
        )

        self.total_counter["text"] = f"Total filtered records:
{self.controller.get_total_count(self.fltr)}"
        for record in self.controller.get_page(self.page, PAGE_SIZE,
self.fltr):
            self.table.insert("", "end", text=record[0], values=record[1:])

    # UI:

    def __make_table(self):
        self.table = ttk.Treeview(self.root, columns=("Состав", "Позиция",
"Титулы", "Вид спорта", "Разряд"))
        self.table.heading("#0", text="ФИО")
        self.table.heading("Состав", text="Состав")
        self.table.heading("Позиция", text="Позиция")
        self.table.heading("Титулы", text="Титулы")
        self.table.heading("Вид спорта", text="Вид спорта")
        self.table.heading("Разряд", text="Разряд")
        self.table.pack()

    def __make_paging_controls(self):
        self.prev_page_btn = ttk.Button(self.root, text="Previous page",
command=self.__handle_prev_page)
        self.prev_page_btn.pack(side="left")
        self.next_page_btn = ttk.Button(self.root, text="Next page",
command=self.__handle_next_page)
        self.next_page_btn.pack(side="left")
        self.total_counter = ttk.Label(text="Total entries found: 0")
        self.total_counter.pack(side="left")
```

```
    def __make_record_controls(self):
        self.remove_all_btn = ttk.Button(self.root, text="Remove all
filtered", command=self.__handle_remove_all)
        self.remove_all_btn.pack(side="right")
        self.remove_record_btn = ttk.Button(self.root, text="Remove record",
command=self.__handle_remove_record)
        self.remove_record_btn.pack(side="right")
        self.add_record_btn = ttk.Button(self.root, text="Add record",
command=self.__handle_add_record)
        self.add_record_btn.pack(side="right")

    def __make_filter_controls(self):
        ttk.Label(self.root, text="Name filter:").pack()
        self.name_filter = ttk.Entry(self.root)
        self.name_filter.bind("<KeyRelease>", self.__render_table)
        self.name_filter.pack()

        ttk.Label(self.root, text="Sport filter:").pack()
        self.sport_filter = ttk.Combobox(self.root)
        self.sport_filter.bind("<<ComboboxSelected>>", self.__render_table)
        self.sport_filter.pack()

        ttk.Label(self.root, text="Rank filter:").pack()
        self.rank_filter = ttk.Combobox(self.root)
        self.rank_filter.bind("<<ComboboxSelected>>", self.__render_table)
        self.rank_filter.pack()

        ttk.Label(self.root, text="Title filter:").pack()
        self.title_filter = ttk.Entry(self.root)
        self.title_filter.bind("<KeyRelease>", self.__render_table)
        self.title_filter.pack()

    # Handlers:

    def __handle_next_page(self):
        self.page += 1
        self.__render_table()

    def __handle_prev_page(self):
        if self.page > 0:
            self.page -= 1
        self.__render_table()

    def __handle_add_record(self):
        AddRecordDialog(self.root, self.__handle_add_record_submit)

    def __handle_remove_record(self):
        selected_items = self.table.selection()
        for selection in selected_items:
            item_id = selection
            value = self.table.item(item_id)['text']
            self.controller.remove_record(value)
```

```
        self.__render_table()

    def __handle_remove_all(self):
        selected_items = self.controller.get_page(self.page, PAGE_SIZE,
self.fltr)
        for item in selected_items:
            self.controller.remove_record(item[0])
        self.__render_table()
        showinfo(message=f"Deleted {len(selected_items)} records.")

    def __handle_add_record_submit(self, dialog):
        name = dialog.name.get()
        squad = dialog.squad.get()
        position = dialog.position.get()
        titles = [x.strip() for x in dialog.title.get().split(",")]
        sport = dialog.sport.get()
        rank = dialog.rank.get()
        self.controller.add_record(name, squad, position, titles, sport, rank)
        self.__render_table()

    def __init__(self, controller: RecordController):
        self.page = 0
        self.controller = controller
        self.root = tk.Tk()

        self.__make_filter_controls()
        self.__make_table()
        self.__make_paging_controls()
        self.__make_record_controls()
        self.__render_table()

    def start(self):
        self.root.mainloop()
```

AddRecordDialog

Данный код представляет класс AddRecordDialog, который создает диалоговое окно для добавления новой записи в приложение. Класс использует библиотеку tkinter для создания графического интерфейса.

Конструктор класса принимает два аргумента: parent - родительское окно, в котором будет отображаться диалоговое окно, и on_submit - функцию обратного вызова, которая будет вызвана при отправке формы.

В методе __init__ класса создаются шесть Entry виджетов для ввода строковых значений, которые соответствуют полям класса Record. Далее, методом grid виджеты добавляются на диалоговое окно в таблицу с двумя колонками и шестью строками.

Для каждого Entry виджета создается соответствующая метка с описанием поля. Методом grid метки также добавляются на диалоговое окно.

Наконец, на диалоговом окне добавляется кнопка "Submit", которая вызывает функцию on_submit, передавая в качестве аргумента сам объект AddRecordDialog. При нажатии на эту кнопку, все поля Entry виджетов будут переданы в обработчик формы, который будет создан в месте, где будет использоваться данный класс.

```
import tkinter as tk


class AddRecordDialog:
   def __init__(self, parent, on_submit):
       # Create a new window
       self.top = tk.Toplevel(parent)
       self.top.title("Dialog")

       # Create 6 Entry widgets for string input
       self.name = tk.Entry(self.top)
       self.squad = tk.Entry(self.top)
       self.position = tk.Entry(self.top)
       self.title = tk.Entry(self.top)
       self.sport = tk.Entry(self.top)
       self.rank = tk.Entry(self.top)

       # Add the Entry widgets to the window using grid layout
       self.name.grid(row=0, column=1)
       self.squad.grid(row=1, column=1)
       self.position.grid(row=2, column=1)
       self.title.grid(row=3, column=1)
       self.sport.grid(row=4, column=1)
       self.rank.grid(row=5, column=1)

       # Add labels to describe each Entry widget
       tk.Label(self.top, text="ФИО").grid(row=0, column=0)
       tk.Label(self.top, text="Состав").grid(row=1, column=0)
       tk.Label(self.top, text="Позиция").grid(row=2, column=0)
       tk.Label(self.top, text="Титулы (через зпт)").grid(row=3, column=0)
       tk.Label(self.top, text="Вид спорта").grid(row=4, column=0)
       tk.Label(self.top, text="Разряд").grid(row=5, column=0)

       # Add a button to submit the inputs
       tk.Button(self.top, text="Submit", command=lambda:
on_submit(self)).grid(row=6, column=1)
```