МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Вычислительной техники

ОТЧЕТ

по Курсовой работе

по дисциплине «Технология объектно-ориентированного программирования и проектирования»

Студент гр. 4312 Устинов В. А.

Преподаватель Лазарев Э. А.

Санкт-Петербург

2025

1 Техническое задание

1.1 Введение

- *Наименование*: программный комплекс "Dog Festival" для управления данными выставки собак;
- *Область применения*: проведение и администрирование выставок собак;
- *Объект использования*: Кинологические клубы, организаторы выставок собак;

1.2 Основания для разработки

Тема разработки: "Программный комплекс для учета и управления данными выставки собак".

1.3 Назначение разработки

- *Функциональное назначение*: автоматизация процессов учета участников выставки собак, обработки данных о собаках;

- *Эксплуатационное назначение*: использование администраторами выставок для ведения базы данных, формирования отчетов, поиска информации.

1.4. Требования к программе

1.4.1. Требования к функциональным характеристикам:

- Хранение данных о собаках (кличка, порода, наличие наград);
- Добавление, редактирование, удаление записей;
- Поиск данных по различным критериям;
- Генерация отчетов в форматах PDF и HTML;
- Сохранение и восстановление данных из XML-файлов.

1.4.2. Требования к надежности:

- Проверка корректности вводимых данных;
- Резервное копирование данных;
- Восстановление данных при сбоях;
- Контроль целостности данных.

1.4.3. Условия эксплуатации:

- Операционная система: Windows 10 и выше;
- Java Runtime Environment 21 и выше;
- Разрешение экрана: 1024×768 и выше;

- Формат данных: XML;
- Язык программирования: Java;
- Библиотеки: Swing для GUI.

1.5. Требования к программной документации

1 Техническое задание;
2 Руководство оператора;
3 Исходный код программы;
4 UML-диаграммы.

1.6. Стадии и этапы разработки

1 Анализ требований (1 неделя)
2 Проектирование (2 недели)
3 Реализация (3 недели)
4 Тестирование (1 неделя)
5 Документирование (1 неделя)

1.7. Порядок контроля и приемки

- Модульное тестирование
- Интеграционное тестирование
- Приемочные испытания

2 Описание вариантов использования ПК

2.1 Акторы системы

В разрабатываемом программном комплексе выделены следующие акторы:

- Пользователь — основной актор системы. Представляет администратора выставки собак, который осуществляет ввод, редактирование, удаление и анализ данных;
- Файловая система — внешний по отношению к ПК компонент, используемый для хранения и загрузки данных в формате XML;
- Подсистема отчетов (*JasperReports*) — внешняя программная система, применяемая для формирования отчетов в различных форматах.

Основным инициатором всех сценариев является пользователь, который взаимодействует с системой через графический интерфейс.

2.2 Основные варианты использования

Совокупность функций программного комплекса представлена следующими прецедентами.

1)Прецедент «Добавляет запись о собаке»
Актор: пользователь
Цель: ввод новой информации о собаке в систему
Предусловие: открыто главное окно приложения.
Основной сценарий:
1. Пользователь нажимает кнопку добавления записи.
2. Система открывает диалоговое окно ввода данных.
3. Пользователь вводит имя собаки, породу и информацию о наградах.
4. Система выполняет валидацию введенных данных.
5. При успешной проверке данные добавляются в структуру List<Dog>.
6. Таблица данных обновляется.

Постусловие: новая запись отображается в таблице и сохранена в памяти приложения.

2)Прецедент «Редактирует запись о собаке»
Актор: пользователь
Цель: изменение существующей информации
Предусловие: в таблице присутствует хотя бы одна запись.
Основной сценарий:
1. Пользователь выбирает запись и нажимает кнопку редактирования.
2. Система отображает окно с текущими значениями полей.
3. Пользователь изменяет необходимые данные.
4. Система выполняет повторную валидацию.
5. Данные в List<Dog> и таблице обновляются.

Постусловие: измененная запись сохранена и отображена в интерфейсе.

3)Прецедент «Удаляет запись о собаке»
Актор: пользователь
Цель: удаление ненужной записи
Предусловие: в системе существуют записи.
Основной сценарий:
1. Пользователь нажимает кнопку удаления.
2. Система запрашивает подтверждение операции.
3. После подтверждения запись удаляется.
4. Таблица перенумеровывается и обновляется.

Постусловие: выбранная запись удалена из системы.

4)Прецедент «Сохраняет данные в файл»
Актор: пользователь
Цель: сохранение текущего состояния данных
Основной сценарий:
1. Пользователь нажимает кнопку сохранения.
2. Система передает данные менеджеру файлов.
3. В отдельном потоке создается XML-файл dogs.xml.
4. Отображается сообщение об успешном завершении операции.

Постусловие: данные сохранены в XML-файл

5)Прецедент «Загружает данные из файла»
Актор: пользователь
Цель: восстановление ранее сохраненных данных
Основной сценарий:
1. Пользователь инициирует загрузку данных.
2. Система считывает XML-файл.
3. Данные загружаются в List<Dog>.
4. Таблица обновляется.

Постусловие: данные успешно восстановлены.

6)Прецедент «Ищет записи»
Актор: пользователь
Цель: быстрый поиск информации
Предусловие: таблица содержит данные.
Основной сценарий:
1. Пользователь вводит поисковый запрос.
2. Нажимает кнопку поиска.
3. Система фильтрует записи по имени или породе.
4. Отображаются только подходящие записи.

Постусловие: выбранная запись найдена.

7)Прецедент «Формирует отчет»
Актор: пользователь
Цель: получение отчетной документации
Основной сценарий:
1. Пользователь выбирает тип отчета.
2. Система формирует отчет с использованием JasperReports.
3. В отчет включается статистическая информация.
4. Отчет сохраняется в файл выбранного формата.

Постусловие: отчет успешно создан и доступен пользователю.

8)Прецедент «Завершает работу программы»
Актор: пользователь
Цель: корректное завершение работы приложения
Основной сценарий:
1. Пользователь нажимает кнопку выхода.
2. Система запрашивает подтверждение.
3. Выполняется автоматическое сохранение данных.

Постусловие: приложение закрывается.

2.3 Связи между прецедентами

В системе используются следующие типы связей UML:

- Связь коммуникации — между пользователем и всеми прецедентами.
- Связь использования (uses) — прецеденты редактирования и удаления используют прецедент валидации данных.
- Связь расширения (extends) — автоматическое сохранение данных расширяет прецедент завершения работы программы.

2.4 UML-Диаграмма

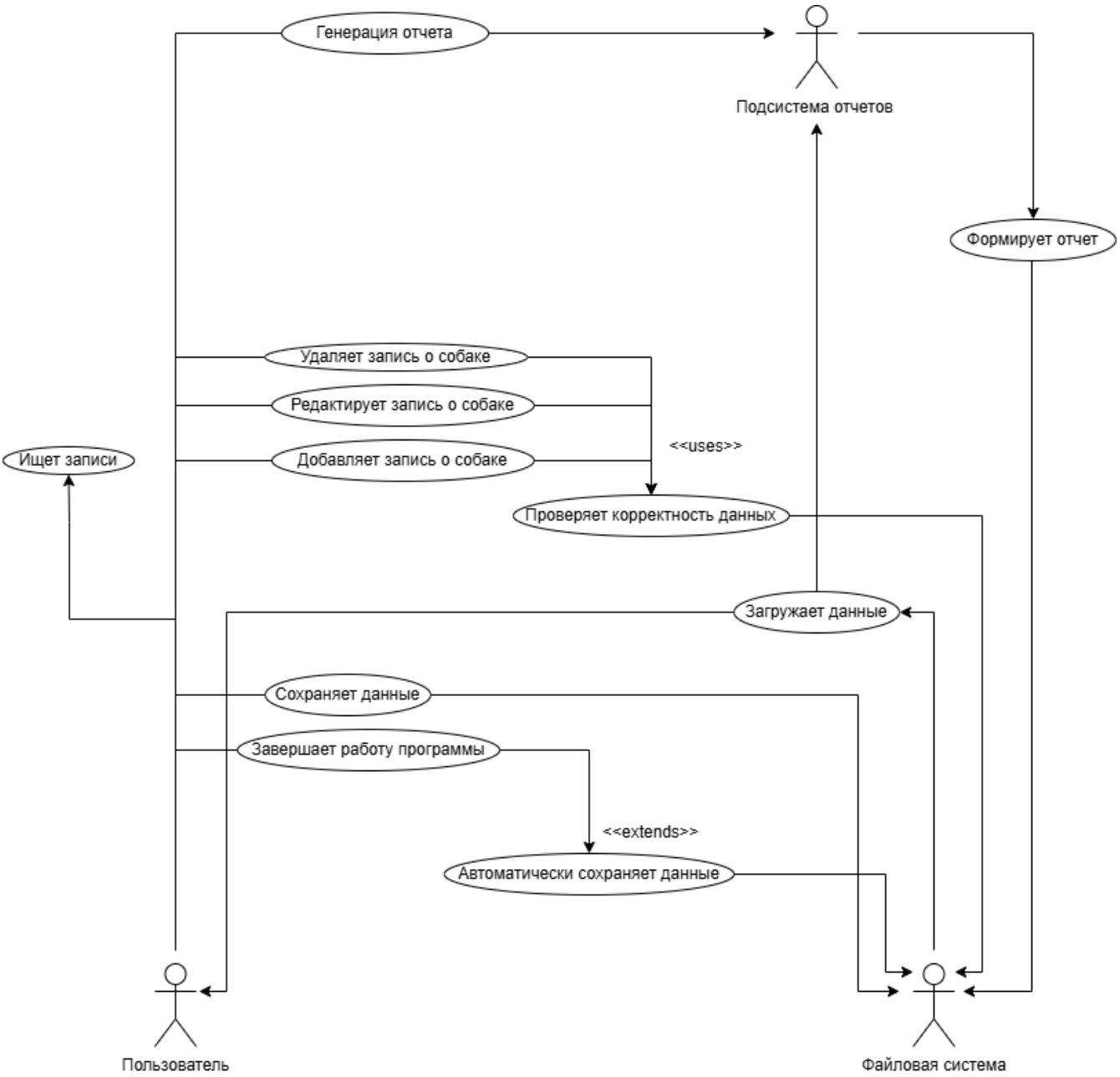

Рисунок 1 — Связи между прецедентами

3 Создание прототипа интерфейса пользователя

3.1 Основная идея прототипа

Пользовательский интерфейс разрабатываемого программного комплекса реализован в виде графического оконного интерфейса и состоит из одной основной экранной формы и набора вспомогательных диалоговых окон, вызываемых из основной формы.

3.2 Экранные формы программного комплекса

В программном комплексе учета собак выставки используются следующие экранные формы:

- «Главное окно программы»;
- «Окно добавления собаки»;
- «Окно редактирования собаки»;
- «Окно удаления собаки»;
- «Окно формирования отчета»;
- «Окно выхода из программы».

3.3 Экранные формы пользовательского интерфейса

Экранная форма	Элементы управления	Действия пользователя	Отклик системы
Главное окно программы	Таблица данных с полями: №, Имя, Порода, Награды. Панель кнопок: «Добавить», «Редактировать», «Удалить», «Сохранить», «Бэк-ап», «Отчет», «Выход». Поле ввода кнопка «Поиск».	Нажать кнопку «Добавить»	Открыть окно «Добавление собаки».
		Нажать кнопку «Редактировать».	Открыть окно «Редактирование собаки» с пред заполненными данными выбранной записи.
		Нажать кнопку «Удалить».	Открыть окно «Удаление собаки».
		Ввести текст в строку поиска и нажать кнопку «Поиск».	Отфильтровать таблицу и отобразить записи, соответствующие запросу.
		Нажать кнопку «Сохранить».	Сохранить данные в XML-файл и вывести сообщение о результате операции.
		Нажать кнопку «Загрузить».	Загрузить данные из XML-файла и обновить таблицу.
		Нажать кнопку «Отчет».	Открыть окно «Формирование отчета».
		Нажать кнопку «Выход».	Открыть окно «Выход из программы».
Окно добавления собаки	Поля ввода: Имя, Порода. Переключатель «Наличие наград». Кнопки:	Ввести данные о собаке и нажать «Сохранить».	Проверить корректность данных, добавить запись в хранилище, обновить таблицу

	«Сохранить», «Отмена».		в главном окне и закрыть форму.
		Нажать кнопку «Отмена».	Закрыть окно без сохранения данных.
Окно редактирования собаки	Поля ввода с текущими значениями: Имя, Порода, Наличие наград. Кнопки: «Сохранить», «Отмена».	Изменить данные и нажать «Сохранить».	Проверить корректность данных, обновить запись в хранилище, обновить таблицу и закрыть форму.
		Нажать кнопку «Отмена».	Закрыть окно без выполнения операции.
Окно удаления собаки	Текст с информацией о выбранной записи. Кнопки: «Удалить», «Отмена».	Нажать кнопку «Удалить».	Удалить запись из хранилища, обновить таблицу и закрыть окно.
		Нажать кнопку «Отмена».	Закрыть окно без выполнения операции.
Окно формирования отчета	Элементы выбора формата отчета (PDF / HTML / оба). Кнопки: «Сформировать», «Отмена».	Выбрать формат и нажать «Сформировать».	Сформировать отчет с использованием JasperReports, сохранить файл и вывести сообщение об успешном завершении.
		Нажать кнопку «Отмена».	Закрыть окно без выполнения операции.
Окно выхода из программы	Текст подтверждения выхода. Кнопки: «Да», «Нет».	Нажать кнопку «Да».	Автоматически сохранить данные и завершить работу программы.
		Нажать кнопку «Нет».	Закрыть окно выхода и

			вернуться в главное окно.

Таблица 1 — описание экранных форм

4 Разработка объектной модели ПК

4.1 Сущности предметной области

В объектной модели программного комплекса учета собак выставки представлены следующие основные сущности:

- Собака;
- Порода;
- Награда;
- Отчет;
- Хранилище данных.

4.2 Описание сущностей и их атрибутов

1)Сущность «Собака»

Атрибуты:

- имя : Строка
- наличие наград : Логическая переменная

2)Сущность «Порода»

Атрибуты:

- название : Строка

3)Сущность «Награда»

Атрибуты:

название : Строка

4)Сущность «Отчет»

Атрибуты:

- тип отчета : Строка
- дата создания : Дата

5)Сущность «Хранилище данных»

Атрибуты:

- Путь к файлу : Строка
- формат : Строка

4.3 Ассоциации между сущностями

Между сущностями объектной модели определены следующие ассоциации:

1 Собака — Имя
Ассоциация: «Относится к»
Кратность:
- одна собака относится ровно к одной породе (1);
- одна порода может быть связана с несколькими собаками (1..*).

2 Собака — Порода
Ассоциация: «Относится к»
Кратность:
- одна собака относится ровно к одной породе (1);
- одна порода может быть связана с несколькими собаками (1..*).

3 Собака — Награда
Ассоциация: «Имеет»
Кратность:
- одна собака может иметь ноль или более наград (0..*);
- каждая награда относится к одной собаке (1).

4 Отчет — Собака
Ассоциация: «Содержит данные о»
Кратность:
- один отчет содержит данные о нескольких собаках (1..*);
- одна собака может входить в несколько отчетов (0..*).

5 Хранилище данных — Собака
Ассоциация: «Хранит»
Кратность:
- хранилище данных хранит множество записей о собаках (0..*).

4.4 Операции сущностей

Сущность	Имя операции	Параметры операции			Тип возвращаемого значения	Назначение операции
		Вид	Название	Тип		
Собака	Создать	Вх.	имя	Строка	Собака	Создает новый объект сущности «Собака» с заданными атрибутами
		Вх.	порода	Строка		
		Вх.	наличие наград	Булево значение		
Собака	Изменить	Вх.	имя	Строка	Пусто	Изменяет данные существующей собаки
		Вх.	порода	Строка		
		Вх.	наличие наград	Булево значение		
Собака	Удалить	Пусто			Пусто	Удаляет объект собаки из хранилища
Хранилище данных	Сохранить	Вх.	список собак	Список собак	Булево значение	Сохраняет данные о собаках в файл
Хранилище данных	Загрузить	Пусто			Список собак	Загружает данные о собаках из файла
Отчет	Сгенерировать	Вх.	список собак	Список собак	Пусто	Назначает награду выбранной собаке
		Вх.	тип отчета	Строка		

Таблица 2 — перечень операций сущностей

4.5 Вывод по объектной модели

Разработанная объектная модель отражает основные понятия предметной области и их взаимосвязи. Выделенные сущности, их атрибуты и операции обеспечивают поддержку всех функциональных требований программного комплекса. Использование UML-диаграммы классов позволяет формализовать объектную структуру системы и служит основой для последующей реализации программного комплекса.

5 Построение диаграммы программных классов

5.1 Описание основных классов системы

Обозначения типа класса:

- + — public;
- - — private;
- # — protected.

1)Класс Dog

Класс Dog соответствует сущности предметной области «Собака» и представляет модель данных.

Атрибуты:

- -name: String — имя собаки
- -poroda: String — порода
- -hasAward: boolean — наличие наград

Методы:

- +getName(): String
- +getBreed(): String
- +hasAward(): boolean
- +setName(name: String): void
- +setBreed(breed: String): void
- +setAwards(hasAward: boolean): void
- +toString(): String

2)Класс List<T>

Класс List<T> реализует собственную структуру односвязного списка и используется для хранения объектов типа Dog.

Атрибуты:

- -head: Node<T> — начало списка
- -end: Node<T> — конец списка
- -sz: int — количество элементов

Методы:

- +push_back(value: T): void
- +push_front(value: T): void
- +insert(index: int, value: T): void
- +remove(index: int): void
- +at(index: int): T
- +replace(index: int, value: T): void
- +convToStr(): String[]

3)FileManager

Класс FileManager предназначен для работы с файловой системой и реализует асинхронные операции загрузки и сохранения данных.

Методы:

- +loadFromXML(callback): void
- +saveToXML(data: List<Dog>, callback): void
- -inputFromXML(): List<Dog>
- -outputToXML(data: List<Dog>): void
- +inputFromCSV(): List<Dog>

- +outputToCSV(data: List<Dog>): void

4)ReportGenerator
Класс ReportGenerator реализует генерацию отчетов с использованием библиотеки JasperReports.
Методы:
- +generatePDFReport(data: List<Dog>): boolean
- +generateHTMLReport(data: List<Dog>): boolean
- -convertToMapList(data: List<Dog>): List<Map>
- -createParams(data: List<Dog>): Map

5)InputException
Класс InputException используется для обработки ошибок пользовательского ввода.
Перечисление:
ErrorType { EMPTY_FIELD, INVALID_NUMBER, OUT_OF_RANGE, INVALID_FORMAT }
Методы:
- +validEmptyField(value: String): void
- +validRowNumber(value: String, max: int): void
- +validLettersOnly(value: String): void
- +validZeroOrOne(value: String): void

6)Абстрактный класс InputOutputWindow
Базовый класс для всех экранных форм приложения.
Методы:
- #getData(): String[]
- #successOperationWindow(message: String): void
- #confirmOperationWindow(message: String): boolean
- #showErrorDialog(message: String): void

7)MainWindow (наследник InputOutputWindow)
Главное окно приложения.
Методы:
- +show(): void
- -handleButtonClick(): void
- -searchElement(query: String): void
- -exitApplication(): void

8)AddElementWindow (наследник InputOutputWindow)
Окно добавления новой записи.
Методы:

- +show(): void
- -showDataAdding(): void
- -addRowToTable(): void

9)EditElementWindow (наследник InputOutputWindow)
Окно редактирования записи.
Методы:
- +show(): void
- -showRowSelection(): void
- -showDataEditing(): void
- -EditRowByNumber(): void

10)DeleteElementWindow (наследник InputOutputWindow)
Окно удаления записи.
Методы:
- +show(): void
- -showDeleteRow(): void
- -deleteRowByNumber(): void
- -updateRowNumbers(): void

11)AddReportWindow (наследник InputOutputWindow)
Окно генерации отчетов.
Методы:
- +show(): void
- -generateReport(): void
- -getData(): String[]

5.2 Отношения между классами

В диаграмме программных классов используются следующие виды связей:
Ассоциация:
- MainWindow ассоциирован с List<Dog>, FileManager, ReportGenerator;
- Экранные формы работают с объектами Dog;

Агрегация:
- MainWindow агрегирует объекты экранных форм;
- List<Dog> агрегирует объекты класса Dog;

Наследование:
- AddElementWindow, EditElementWindow, DeleteElementWindow, AddReportWindow наследуют InputOutputWindow.

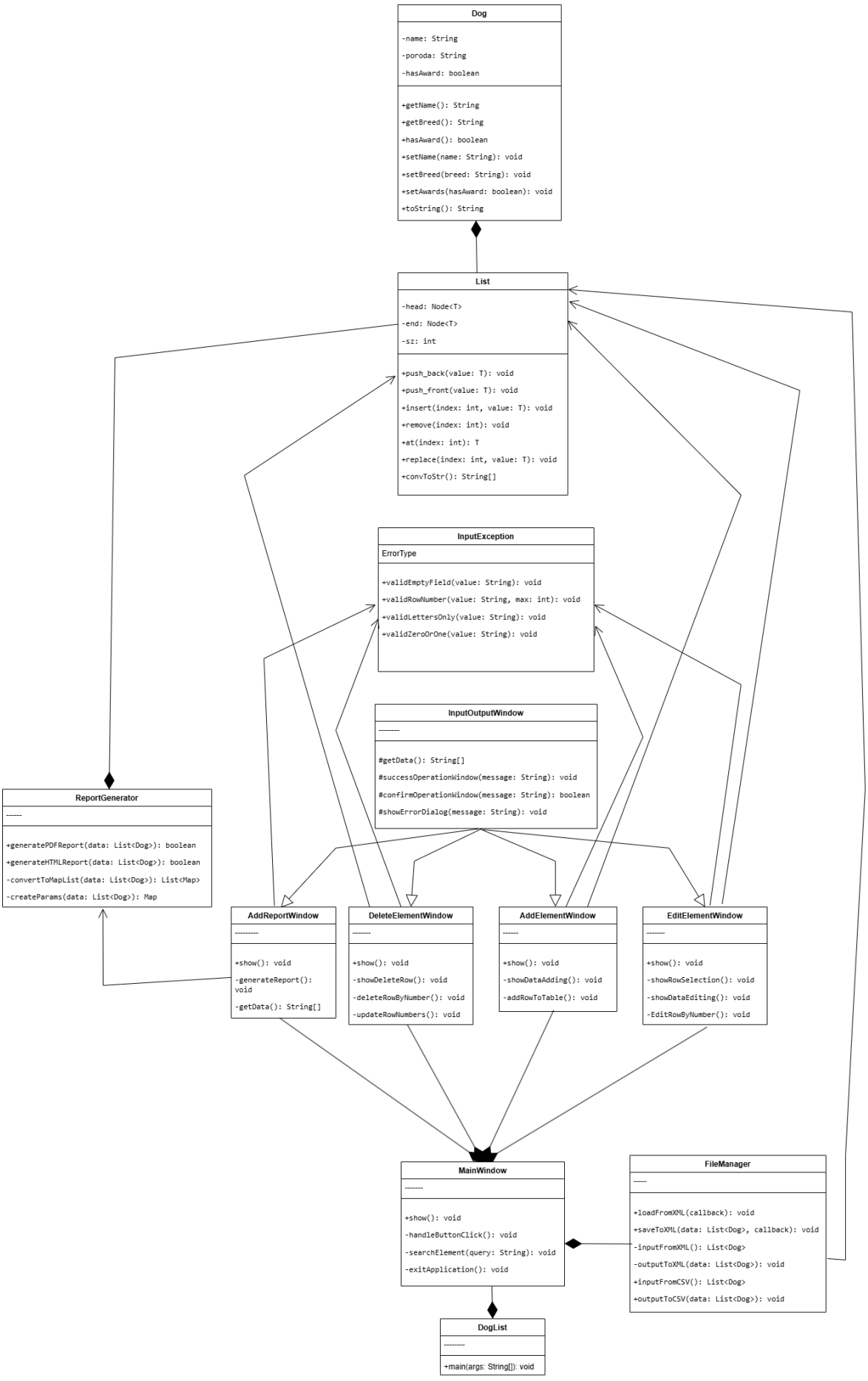

Рисунок 2 — Диаграмма программных классов

6 Описание поведения ПК

6.1 Диаграмма последовательностей сценария «Добавление собаки»

В реализации сценария «Добавление собаки» участвуют следующие объекты:

- User — пользователь системы (администратор выставки);
- MainWindow : MainWindow — главное окно приложения;
- AddElementWindow : AddElementWindow — окно добавления записи;
- Dog : Dog — объект предметной области «Собака»;
- DogList : List<Dog> — хранилище объектов Dog;
- InputException — класс валидации пользовательского ввода.

Все объекты изображаются в верхней части диаграммы, для каждого проводится линия жизни.

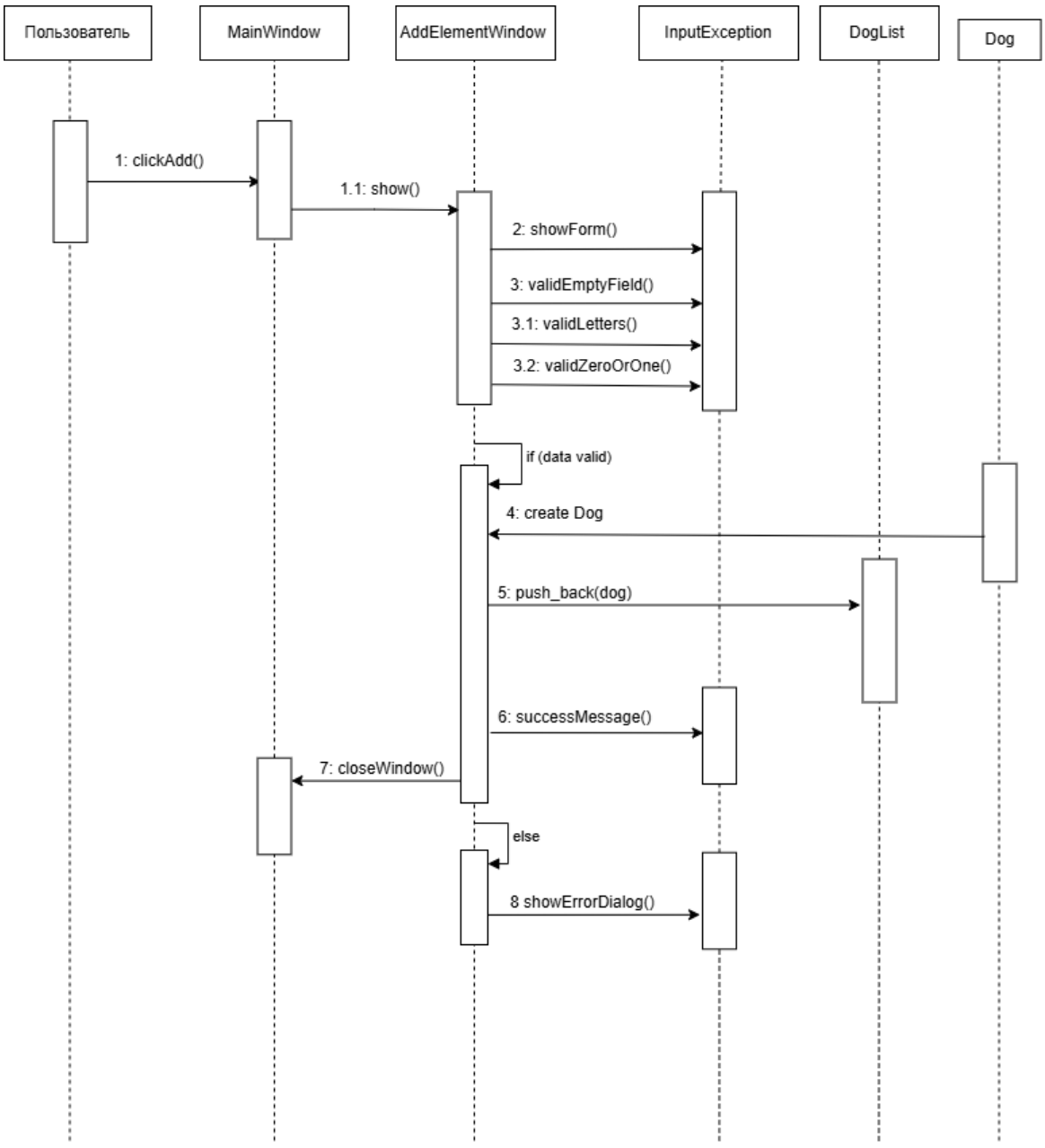

Рисунок 3 — Диаграмма поведения ПК

7 Диаграмма действий

7.1 «Добавление собаки»

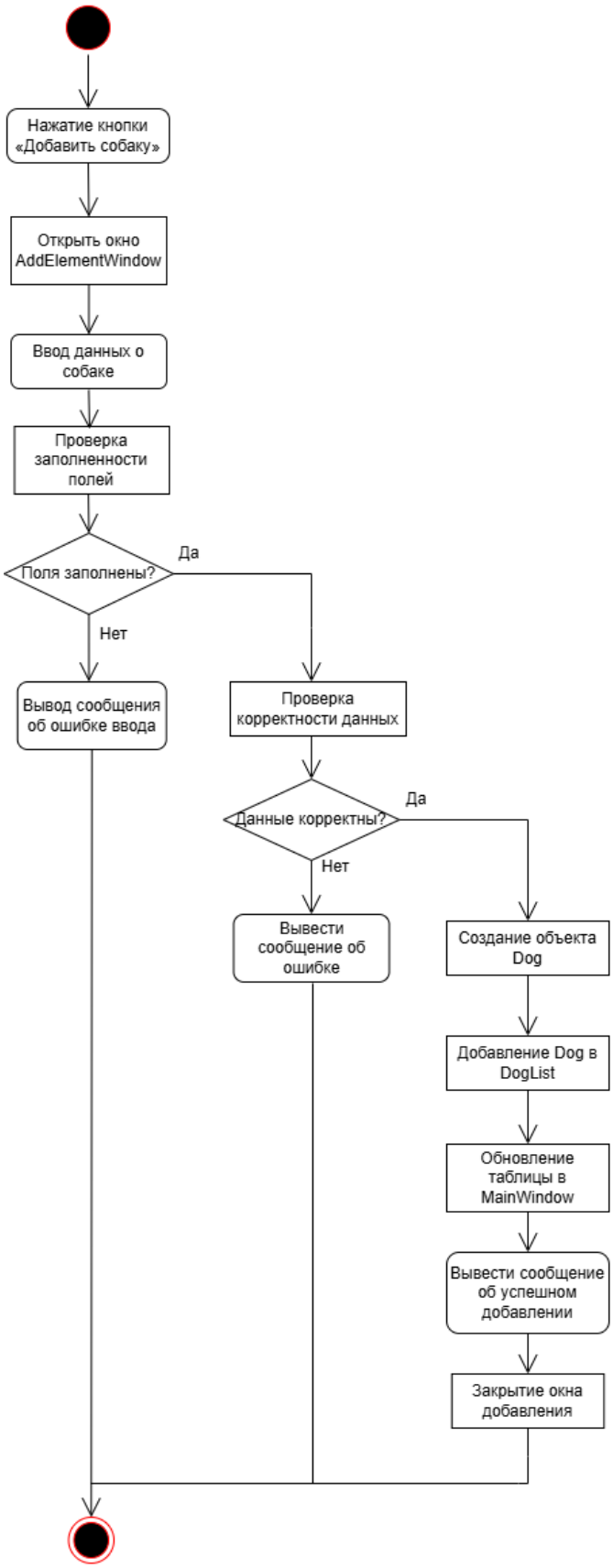

Рисунок 4 — Диаграмма действия AddElementWindow

7.2 «Удаление собаки»

Рисунок 5 — Диаграмма действия DeleteElementWindow

7.3 «Редактирование собаки»

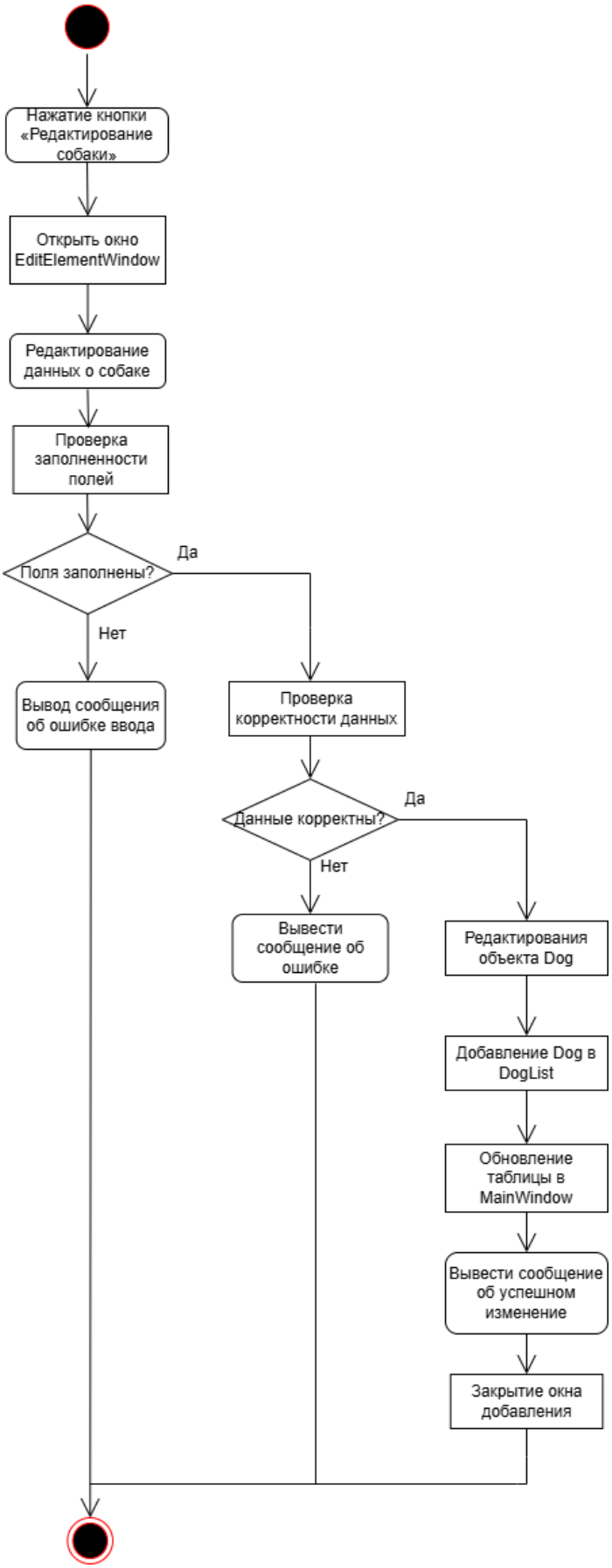

Рисунок 6 — Диаграмма действия EditElementWindow

7.4 «Генерация отчета»

Рисунок 7— Диаграмма действия AddReportWindow

8 Программная документация

8.1 Назначение программы

Программный комплекс «Dog Festival» предназначен для автоматизации учета и обработки данных о собаках, участвующих в выставке собак. Программа позволяет добавлять, редактировать, удалять и просматривать информацию о собаках, выполнять поиск по данным, сохранять и загружать данные из файлов, а также формировать отчеты в форматах PDF и HTML.
Программа ориентирована на использование администраторами выставок собак и не требует специальных навыков программирования.

8.2 Условия выполнения программы

Для корректной работы программы требуется следующее аппаратное и программное обеспечение:

- персональный компьютер с операционной системой Windows / Linux;
- процессор с тактовой частотой не ниже 1.5 ГГц;
- объем оперативной памяти не менее 2 ГБ;
- установленная среда выполнения Java Runtime Environment (JRE) версии 21 и выше;
- доступ к файловой системе для сохранения и загрузки данных;
- установленная библиотека JasperReports (включена в проект).

8.3 Описание задачи

Задачей программного комплекса является обеспечение удобного и надежного управления данными о собаках, участвующих в выставке. Программа решает следующие задачи:

- хранение информации о собаках (имя, порода, наличие наград);
- выполнение операций добавления, редактирования и удаления данных;
- поиск и фильтрация записей;
- сохранение и загрузка данных в формате XML;
- формирование отчетов с использованием JasperReports.

Для решения задачи используется графический пользовательский интерфейс и внутреннее хранилище данных в виде списка объектов.

8.4 Входные и выходные данные

Входные данные:

- данные, вводимые пользователем через экранные формы(имя собаки, порода собаки, информация о наличии наград);
- файлы формата XML и CSV, загружаемые из файловой системы.

Выходные данные:

- таблица с отображением данных о собаках;
- XML и CSV файлы с сохранёнными данными;

- отчеты в форматах PDF и HTML;
- информационные и диагностические сообщения.

8.5 Выполнение программы

Для работы с программой оператор должен выполнить следующие действия:

1. Запустить программу двойным щелчком по исполняемому файлу.
2. После запуска отобразится главное окно программы с таблицей данных.
3. Для добавления новой записи нажать кнопку «Добавить».
4. Для редактирования записи нажать кнопку «Редактировать».
5. Для удаления записи нажать кнопку «Удалить».
6. Для поиска данных использовать строку поиска.
7. Для сохранения или загрузки данных использовать соответствующие кнопки.
8. Для формирования отчетов открыть окно генерации отчетов.
9. Для завершения работы нажать кнопку «Выход».

8.6 Проверка программы

Проверка работоспособности программы осуществляется путем выполнения контрольных примеров:

- добавление новой записи;
- редактирование существующей записи;
- удаление записи;
- сохранение данных в файл и последующая загрузка;
- генерация отчетов.

Результатом успешной проверки является корректное выполнение всех операций без возникновения ошибок.

8.7 Сообщения оператору

В процессе работы программа может выводить следующие сообщения:

- «Данные успешно добавлены» — операция выполнена успешно;
- «Ошибка ввода данных» — введены некорректные данные;
- «Файл не найден» — указан неверный путь к файлу;
- «Отчет успешно сформирован» — отчет создан;
- «Вы уверены, что хотите выйти?» — запрос подтверждения выхода.

В случае возникновения ошибок оператору рекомендуется проверить корректность введенных данных или повторить операцию.

9 Исходные тексты ПК.

9.1 Пакет fileManager

1)Filemanager.java

```
package fileManager;

import list.List;
import object.dog.Dog;
import java.io.IOException;
import java.nio.file.*;
import java.util.Arrays;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.*;
import org.xml.sax.SAXException;

/**
 * Manages file operations with multithreading support.
 */
public class FileManager {

    /**
     * Loads data from XML file asynchronously.
     * @param filePath Path to the XML file
     * @param callback for handling the result
     */
    public void loadFromXML(String filePath, FileOperationCallback<List<Dog>>
callback) {
        Thread loadThread = new Thread(() -> {
            try {
                List<Dog> dogs = inputFromXML(filePath);
                callback.onSuccess(dogs);
            } catch (Exception e) {
                callback.onError(e);
            }
        });
        loadThread.setName("Data-Loader-Thread");
        loadThread.start();
    }

    /**
     * Saves data to XML file asynchronously.
     * @param filePath Path to the XML file
     * @param dogs List of dogs to save
     * @param callback for handling the result
     */
    public void saveToXML(String filePath, List<Dog> dogs, FileOperation-
Callback<Void> callback) {
```

```
        Thread saveThread = new Thread(() -> {
            try {
                outputToXML(filePath, dogs);
                callback.onSuccess(null);
            } catch (Exception e) {
                callback.onError(e);
            }
        });
        saveThread.setName("Data-Saver-Thread");
        saveThread.start();
    }

    /**
     * Saves modified dog data to XML for reporting purposes.
     * @param filePath Path to the XML file
     * @param dogs List of dogs to process
     * @param callback for handling the result
     */
    public void saveModifiedForReport(String filePath, List<Dog> dogs, FileOpera-
tionCallback<Void> callback) {
        Thread saveThread = new Thread(() -> {
            try {
                List<Dog> reportDogs = new List<>();
                for (int i = 0; i < dogs.getSize(); i++) {
                    Dog original = dogs.at(i);
                    Dog modifiedDog = new Dog(
                        "[Report] " + original.getName(),
                        original.getBreed(),
                        original.hasAward()
                    );
                    reportDogs.push_back(modifiedDog);
                }

                outputToXML(filePath, reportDogs);
                callback.onSuccess(null);
            } catch (Exception e) {
                callback.onError(e);
            }
        });
        saveThread.setName("Data-Editor-Thread");
        saveThread.start();
    }

    /**
     * Callback interface for asynchronous file operations.
     * @param <T> Type of operation result
     */
    public interface FileOperationCallback<T> {
        /**
         * Called when operation completes successfully.
```

```
         * @param result Operation result
         */
        void onSuccess(T result);

        /**
         * Called when operation fails.
         * @param e Exception that occurred
         */
        void onError(Exception e);
    }

    /**
     * Reads dog data from XML file.
     * @param filePath Path to the XML file
     * @return List of dogs read from file
     * @throws IOException If an I/O error occurs
     */
    public List<Dog> inputFromXML(String filePath) throws IOException {
        List<Dog> dogList = new List<>();
        try {
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();
            Document doc = builder.parse(Files.newIn-
putStream(Paths.get(filePath)));

            doc.getDocumentElement().normalize();
            NodeList dogNodes = doc.getElementsByTagName("dog");

            for(int i = 0; i < dogNodes.getLength(); i++) {
                Element dogElem = (Element) dogNodes.item(i);
                String name = dogElem.getEle-
mentsByTagName("name").item(0).getTextContent();
                String breed = dogElem.getEle-
mentsByTagName("breed").item(0).getTextContent();
                boolean hasAward = Boolean.parseBoolean(dogElem.getEle-
mentsByTagName("awards").item(0).getTextContent());

                dogList.push_back(new Dog(name, breed, hasAward));
            }
        } catch (ParserConfigurationException | SAXException e) {
            throw new IOException("XML parsing error: " + e.getMessage(), e);
        }
        return dogList;
    }

    /**
     * Writes dog data to XML file.
     * @param filePath Path to the XML file
     * @param dogs List of dogs to write
     * @throws IOException If an I/O error occurs
```

```
     */
    public void outputToXML(String filePath, List<Dog> dogs) throws IOException {
        try {
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();
            Document doc = builder.newDocument();

            Element rootElement = doc.createElement("dogs");
            doc.appendChild(rootElement);

            for (int i = 0; i < dogs.getSize(); i++) {
                Dog = dogs.at(i);
                Element dogElement = doc.createElement("dog");
                rootElement.appendChild(dogElement);

                Element nameElement = doc.createElement("name");
                nameElement.appendChild(doc.createTextNode(dog.getName()));
                dogElement.appendChild(nameElement);

                Element breedElement = doc.createElement("breed");
                breedElement.appendChild(doc.createTextNode(dog.getBreed()));
                dogElement.appendChild(breedElement);

                Element awardsElement = doc.createElement("awards");
                awardsElement.appendChild(doc.createTextNode(String.valueOf(dog.ha-
sAward())));
                dogElement.appendChild(awardsElement);
            }

            TransformerFactory = TransformerFactory.newInstance();
            Transformer = transformerFactory.newTransformer();
            DOMSource source = new DOMSource(doc);
            StreamResult result = new StreamResult(Files.newOut-
putStream(Paths.get(filePath)));
            transformer.transform(source, result);
        } catch (ParserConfigurationException | TransformerException e) {
            throw new IOException("XML writing error: " + e.getMessage(), e);
        }
    }

    /**
     * Reads dog data from CSV file.
     * @param filePath Path to the CSV file
     * @return List of dogs read from file
     * @throws IOException If an I/O error occurs
     */
    public List<Dog> inputFromCSV(String filePath) throws IOException {
        List<Dog> dogList = new List<>();
        java.util.List<String> lines = Files.readAllLines(Paths.get(filePath));
```

```
        for(String line:lines) {
            String[] dogInfo = line.split(";");
            boolean hasAward = Integer.parseInt(dogInfo[2]) == 1;
            dogList.push_back(new Dog(dogInfo[0], dogInfo[1], hasAward));
        }
        return dogList;
    }

    /**
     * Writes dog data to CSV file.
     * @param filePath Path to the CSV file
     * @param dogs List of dogs to write
     * @throws IOException If an I/O error occurs
     */
    public void outputToCSV(String filePath, List<Dog> dogs) throws IOException {
        String[] listString = dogs.convToStr();
        Files.write(Paths.get(filePath), Arrays.asList(listString));
    }
}
```

9.2 Пакет ScreenForms

1)MainWindow.java

```
package ScreenForms;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableColumn;
import javax.swing.table.TableColumnModel;
import javax.swing.table.DefaultTableCellRenderer;

import java.awt.Image;
import java.awt.BorderLayout;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.FlowLayout;

import java.io.IOException;

import fileManager.FileManager;
import list.List;
import object.dog.Dog;

/**
 * Main window class for the Dog Festival application
 * Handles GUI creation, user interface components, and application workflow
 * Provides functionality for displaying dog data, adding, editing, deleting rec-
ords,
 * searching, and other table operations
 * @author Vadim Ustinov
 * @version 2.0
```

```
 */
public class MainWindow
{
    private static List<Dog> dogs;
    private static FileManager fileMngr;
    private ImageIcon icon;
    private String[][] tableData;
    private JPanel buttonsPanel;
    private JPanel inputPanel;
    private JButton[] buttons;
    private JScrollPane tableScrollPane;
    private static JFrame mainFrame;
    private static JTextField searchTextField;
    private static String[][] originalTableData;
    private static DefaultTableModel tableModel;
    private static JTable dogsTable;
    private static Font defaultFont;
    private static Font headerFont;

    private static final String DOGS_FILE_PATH = "src/data/dogs.xml";

    /** Array of tooltips for application buttons */
    private static final String[] tooltips = {
            "Save",
            "Open",
            "Backup",
            "Add",
            "Remove",
            "Edit",
            "Print",
            "Dropout",
            "Search"
    };

    /** Array of image paths for button icons and application images */
    private static final String[] imagePaths = {
            "src/picts/save.png",
            "src/picts/folder_documents.png",
            "src/picts/cloud.png",
            "src/picts/plus.png",
            "src/picts/minus.png",
            "src/picts/edit.png",
            "src/picts/print.png",
            "src/picts/exit.png",
            "src/picts/search.png",
            "src/picts/dogIcon.png",
            "src/picts/exit.jpg"
    };

    /** Array of column names for the data table */
```

```
    private static final String[] columnNames = {
            "№",
            "Name",
            "Breed",
            "Awards"
     };

    /**
     * Makes the main application window visible
     * Displays the initialized GUI components to the user
     */
    public void show()
    {
        mainFrame.setVisible(true);
    }

    /**
     * Constructs the main application window and initializes all components
     * Loads data from file, creates GUI elements, and sets up event handlers
     * @throws InputException if there's an error reading data files or initializ-
ing components
     */
    public MainWindow() throws InputException
    {
        fileMngr = new FileManager();//init FileManager object
        dogs = new List<>();//init List for dogs data

        //DATA INIT SECTION
        initData();

        //STYLE SECTION
        initMainFrame();
        initFonts();

        //BUTTON SECTION
        initButtonsPanel();

        //TABLE SECTION
        initTable();

        //SEARCH SECTION
        initSearchPanel();

        //ASSEMBLE MAINFRAME SECTION
        assembleMainFrame();
    }

    /**
     * Initializes application data by loading dog information from XML file
     * Creates FileManager and List objects to store and manage dog data
```

```
     * @throws InputException if XML file cannot be read or contains invalid data
     */
    private void initData() throws InputException
    {
        try {
            dogs = fileMngr.inputFromXML(DOGS_FILE_PATH); //writes data from
dogs3.XML
        } catch (Exception e) {
            throw new InputException("Error initializing data: " + e.getMessage(),
                                     InputException.ErrorType.INVALID_FORMAT);
        }
    }

    /**
     * Initializes font settings for the application
     * Sets default font for regular text and header font for table headers
     */
    private void initFonts()
    {
        defaultFont = new Font("Arial", Font.PLAIN, 14);
        headerFont = new Font("Arial", Font.BOLD, 16);
    }

    /**
     * Initializes the main application frame
     * Sets window title, icon, size, and default close operation
     */
    private void initMainFrame()
    {
        icon = new ImageIcon(imagePaths[9]);//init window icon
        mainFrame = new JFrame("Dog Festival");//init mainframe and init window ti-
tle
        mainFrame.setIconImage(icon.getImage());
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainFrame.setSize(900, 500);
        mainFrame.setLocationRelativeTo(null);
    }

    /**
     * Initializes the buttons panel with application control buttons
     * Creates buttons with icons, tooltips, and event handlers
     * Buttons include Save, Open, Backup, Add, Remove, Edit, Print, Dropout, and
Search
     */
    private void initButtonsPanel()
    {
        buttonsPanel = new JPanel();//init buttons panel
        buttonsPanel.setLayout(new GridLayout(1, 8, 10, 10));//set 1 row, 8 cols,
10 width, 10 height
```

```
        buttons = new JButton[9];//init buttons array
        for(int i = 0; i < 9; i++)
        {
            ImageIcon imageForButton = new ImageIcon(imagePaths[i]);//init image
for icon
            Image scaledImage = imageForButton.getImage().getScaledInstance(32, 32,
Image.SCALE_SMOOTH);//scale without loss of quality and fixed size of 32 by 32 px
            ImageIcon buttonIcon = new ImageIcon(scaledImage);//init icon for but-
ton
            final int buttonIndex = i; // need final for lambda function

            buttons[i] = new JButton(buttonIcon);//set icon
            buttons[i].setToolTipText(tooltips[i]);//set button tooltips
            buttons[i].setBorderPainted(false);//remove the frame
            buttons[i].setContentAreaFilled(false);//set transparent background
            buttons[i].setFocusPainted(false);//Remove backlight when focusing
            buttons[i].addActionListener(e -> handleButtonClick(buttonIndex));//add
click handler calls lambda that calls click logic
            if (i < 8) {
                buttonsPanel.add(buttons[i]);//add button into buttonsPanel
            }
        }
    }

    /**
     * Initializes the data table with dog information
     * Converts dog data from List to table format, sets up table model,
     * configures column properties and table appearance
     */
    private void initTable()
    {
        tableData = new String[dogs.getSize()][4];//init table data
        String[] strListDogs = dogs.convToStr();//get dogs data

        // Process each dog's data for display
        for(int i = 0; i < dogs.getSize(); i++)
        {
            String[] RowData = strListDogs[i].split(";");//split separator ;
            tableData[i][0] = String.valueOf(i + 1); // Row number
            tableData[i][1] = RowData[0]; // Dog name
            tableData[i][2] = RowData[1]; // Dog breed
            tableData[i][3] = RowData[2]; // Dog awards
        }

        originalTableData = new String[tableData.length][4];//init reserve copy ta-
ble data
        for (int i = 0; i < tableData.length; i++) {
            System.arraycopy(tableData[i], 0, originalTableData[i], 0, 4);//copy
array (source_arr, start_i, target_arr, start_i, length_of_elements)
        }
```

```
        // Create table model
        tableModel = new DefaultTableModel(tableData, columnNames);//init table
model
        dogsTable = new JTable(tableModel);//creating a data visualization
        dogsTable.setDefaultEditor(Object.class, null);//prohibits editing of table
cells

        // Configure first column
        TableColumnModel columnModel = dogsTable.getColumnModel();
        TableColumn numberColumn = columnModel.getColumn(0);
        numberColumn.setMaxWidth(35);

        // Center align row numbers
        DefaultTableCellRenderer centerRenderer = new DefaultTableCellRenderer();
//creating a render for column
        centerRenderer.setHorizontalAlignment(JLabel.CENTER); //Adjusting the text
alignment of the render
        numberColumn.setCellRenderer(centerRenderer); //Applying a renderer to a
column

        // Configure table appearance
        dogsTable.setFont(defaultFont);
        dogsTable.getTableHeader().setFont(headerFont);
        dogsTable.getTableHeader().setReorderingAllowed(false); //prohibition to
move columns
        dogsTable.setRowHeight(25);

        tableScrollPane = new JScrollPane(dogsTable);
    }

    /**
     * Initializes the search panel with text field and search button
     * Provides functionality for filtering table data based on user input
     */
    private void initSearchPanel()
    {
        inputPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 10, 5));//init
text panel for search
        searchTextField = new JTextField(12);
        searchTextField.setFont(defaultFont);
        buttons[8].addActionListener(e -> handleButtonClick(8));
        inputPanel.add(searchTextField);
        inputPanel.add(buttons[8]);
    }

    /**
     * Assembles all GUI components into the main application frame
     * Arranges buttons panel, table, and search panel in appropriate layout posi-
tions
```

```
     */
    private void assembleMainFrame()
    {
        mainFrame.add(buttonsPanel, BorderLayout.NORTH);
        mainFrame.add(tableScrollPane, BorderLayout.CENTER);
        mainFrame.add(inputPanel, BorderLayout.SOUTH);
    }

    /**
     * Handles all button clicks in the application
     * Routes button events to appropriate functionality based on button index
     * @param buttonIndex the index of the clicked button (0-8)
     */
    private static void handleButtonClick(int buttonIndex)
    {
        if (buttonIndex < tooltips.length)
        {
            try {
                switch(buttonIndex)
                {
                    case 0:
                        try
                        {
                            fileMngr.outputToXML(DOGS_FILE_PATH, dogs);
                            JLabel message = new JLabel("Data saved success-
fully!");
                            message.setFont(new Font("Arial", Font.PLAIN, 16));
                            message.setHorizontalAlignment(SwingConstants.CENTER);
                            JOptionPane.showMessageDialog(
                                    mainFrame,
                                    message,
                                    "Save",
                                    JOptionPane.PLAIN_MESSAGE
                                );
                        }
                        catch (IOException e)
                        {
                            //e.printStackTrace();
                        }
                        break;
                    case 2:
                        try
                        {
                            dogs = fileMngr.inputFromXML(DOGS_FILE_PATH);
                            updateTableWithDataDogs(dogs);
                            JLabel message = new JLabel("Date restored success-
fully!");
                            message.setFont(new Font("Arial", Font.PLAIN, 16));
                            message.setHorizontalAlignment(SwingConstants.CENTER);
                            JOptionPane.showMessageDialog(
```

```
                                    mainFrame,
                                    message,
                                    "Backup",
                                    JOptionPane.PLAIN_MESSAGE
                                );

                        }
                        catch (IOException e)
                        {
                            //e.printStackTrace();
                        }
                        break;
                    case 3:
                        AddElementWindow addElem = new AddElementWindow(dogsTable,
dogs);
                        addElem.show();
                        break;
                    case 4:
                        DeleteElementWindow deleteWindow = new DeleteElementWin-
dow(dogsTable, dogs);
                        deleteWindow.show();
                        break;
                    case 5:
                        EditElementWindow editWindow = new EditElementWin-
dow(dogsTable, dogs);
                        editWindow.show();
                        break;
                    case 6:
                        AddReportWindow reportWindow = new AddReportWindow();
                        reportWindow.show();
                        break;
                    case 7:
                        exitApplication();
                        break;
                    case 8:
                        String text = searchTextField.getText();
                        searchElement(text);
                        break;
                }
            } catch (InputException e) {
                JOptionPane.showMessageDialog(
                    mainFrame,
                    e.getMessage(),
                    "Error",
                    JOptionPane.ERROR_MESSAGE
                );
            }
        }
    }
```

```
    /**
     * Displays a confirmation dialog box with exit image and confirmation text
     * Prompts user to confirm application exit before terminating the program
     * @throws IOException
     */
    private static void exitApplication()
    {
        JPanel exitPanel = new JPanel(new BorderLayout());

        ImageIcon exitImage = new ImageIcon(imagePaths[10]);
        Image scaledImage = exitImage.getImage().getScaledInstance(200, 200, Im-
age.SCALE_SMOOTH);
        JLabel imageLabel = new JLabel(new ImageIcon(scaledImage));
        imageLabel.setHorizontalAlignment(JLabel.CENTER);

        JLabel textLabel = new JLabel("Are you sure you want to exit?", JLabel.CEN-
TER);
        textLabel.setFont(defaultFont);

        exitPanel.add(imageLabel, BorderLayout.CENTER);
        exitPanel.add(textLabel, BorderLayout.SOUTH);

        UIManager.put("OptionPane.buttonFont", defaultFont);

        int result = JOptionPane.showConfirmDialog(
            mainFrame, //parent window
            exitPanel,  //message
            "Confirm Exit", //window title
            JOptionPane.YES_NO_OPTION, //type of buttons
            JOptionPane.PLAIN_MESSAGE //type of message
        );

        if (result == JOptionPane.YES_OPTION)
        {
            try {
                fileMngr.outputToXML(DOGS_FILE_PATH, dogs);
            } catch (IOException e) {
                e.printStackTrace();
            }
            System.exit(0);
        }
    }

    /**
     * Searches the table and shows only the matching rows
     * Filters table data based on search text, showing rows where any cell starts
with the search text
     * @param searchText the text to search for in table cells
     */
    private static void searchElement(String searchText)
```

```
{
    // If no search text, show all data again
    if (searchText == null)
    {
        restoreData();
        return;
    }
    // Make search text lowercase and remove extra spaces
    searchText = searchText.toLowerCase().trim();

    // Create a list to store matching rows
    List<String[]> findedData = new List<>();

    // Look through every row in the table
    for (int i = 0; i < originalTableData.length; i++)
    {
        String[] row = originalTableData[i];
            boolean found = false;
            // Check every cell in this row
            for(String cell : row)
            {
                // If cell starts with search text, we found a match
                if (cell.toLowerCase().startsWith(searchText))
                {
                    found = true;
                    break; // Stop checking this row
                }
            }
            // If we found matching text, save this row
            if(found)
            {
                findedData.push_back(row);
            }
    }
    updateTableWithData(findedData);
}

/**
 * Shows all the data again after searching
 * Restores the original full table data, clearing any search filters
 */
private static void restoreData()
{
    // Clear the table
    tableModel.setRowCount(0);
    // Add back all the original rows
    for (String[] row : originalTableData)
    {
        tableModel.addRow(row);
    }
```

```
    }

    /**
     * Updates the table to show only certain rows
     * Replaces current table data with the provided filtered data set
     * @param data the rows to display in the table after filtering
     */
    private static void updateTableWithData(List<String[]> data)
    {
        // Clear the table
        tableModel.setRowCount(0);
        // Add each row from the search results
        for (int i = 0; i < data.getSize(); i++)
        {
            String[] row = data.at(i);
            tableModel.addRow(row);
        }
    }

    private static void updateTableWithDataDogs(List<Dog> data)
    {
        // Clear the table
        tableModel.setRowCount(0);
        String[] dogData = new String[3];
        String[] row = new String[4];
        // Add each row from the search results
        for (int i = 0; i < data.getSize(); i++)
        {
            dogData = data.at(i).toString().split(";");
            row[0] = Integer.toString(i+1);
            for(int j=0;j<3;j++)
            {
                row[1+j] = dogData[j];
            }
            tableModel.addRow(row);
        }
    }

}
```

2)AddElementWindow.java

```
package ScreenForms;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import list.List;
import object.dog.Dog;


/**
```

```
 * Window for adding new rows to the table
 * User can enter data for a new dog entry
 * @author Vadim Ustinov
 * @version 1.1
 */
public class AddElementWindow extends InputOutputWindow
{
    private JTable addTable;
    private List<Dog> dogs;
    private DefaultTableModel tableModel;
    private boolean operationCancelled;

    private static final String[] FIELD_NAMES = {"Name", "Breed", "Awards"};

    /**
     * Creates window for selecting position to add new row
     * @param table the table to add new row to
     * @param _dogs the list of dogs to modify
     */
    public AddElementWindow(JTable table, List<Dog> _dogs)
    {
        super("Enter the data to add (1 - Name; 2 - Breed; 3 - Awards)", "Add Row",
2, 1);
        addTable = table;
        dogs = _dogs;
        tableModel = (DefaultTableModel) addTable.getModel();
    }

    /**
     * Shows the add window and processes new data
     * Gets user input and adds new row to table
     */
    public void show() throws InputException
    {
        showDataAdding();
    }

    /**
     * Enter data for new row
     * Shows error dialog and retries on invalid data input
     */
    private void showDataAdding() throws InputException
    {
        boolean validInput = false;
        operationCancelled = false;
        while (!validInput && !operationCancelled)
        {
            IODialog.setVisible(true);
            String[] addData = getData();
```

```
            if (addData != null)
            {
                InputException.validEmptyField(textFields, FIELD_NAMES);
                InputException.validDataArray(addData, 3);
                InputException.validLettersOnly(addData[0], FIELD_NAMES[0]);
                InputException.validLettersOnly(addData[1], FIELD_NAMES[1]);
                InputException.validZeroOrOne(addData[2], FIELD_NAMES[2]);

                addRowToTable(addData);

                if (SCSDialog != null)
                {
                    SCSDialog.setVisible(true);
                }

                validInput = true;
            }
            else
            {
                operationCancelled = true;
            }
        }
    }

    /**
     * Adds a new row to the table at specified position
     * @param rowToAdd position to insert new row (1-based)
     * @param newData the data to add [name, breed, awards]
     */
    private void addRowToTable(String[] newData)
    {
        boolean hasAwards=false;
        if(newData[2]=="1")
        {
            hasAwards=true;
        }
        Dog newDog = new Dog(newData[0], newData[1], hasAwards);
        dogs.push_back(newDog);
        int newRowNumber = tableModel.getRowCount() + 1;
        tableModel.addRow(new Object[]{
                String.valueOf(newRowNumber),        // Row number
                newData[0].trim(),   // Dog name
                newData[1].trim(),    // Dog breed
                newData[2].trim()     // Dog awards
            });

        successOperationWindow("Row added");
    }

}
```

3)AddReportWindow.java

```java
package ScreenForms;

import javax.swing.*;
import java.awt.*;

/**
 * Window for generating reports in PDF and HTML formats
 */
public class AddReportWindow extends InputOutputWindow
{
    private JPanel mainPanel;
    private ButtonGroup RadioButtonsGroup;
    private JPanel radioButtonsPanel;
    private JRadioButton pdfRadio;
    private JRadioButton htmlRadio;
    private JRadioButton bothRadio;
    private static final String XML_FILE_PATH = "src/data/dogs.xml";

    /**
     * Creates window for report generation
     */
    public AddReportWindow()
    {
        super("Select report type", "Generate Report", 0, 0);

        //Create panel for options with padding
        mainPanel = new JPanel();
        mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y_AXIS));
        mainPanel.setBorder(BorderFactory.createEmptyBorder(20, 30, 20, 30));

        //Radio buttons for report type selection
        radioButtonsPanel = new JPanel(new GridLayout(3, 1, 10, 10));
        radioButtonsPanel.setMaximumSize(new Dimension(400, 100));

        RadioButtonsGroup= new ButtonGroup();
        pdfRadio = new JRadioButton("PDF Report");
        htmlRadio = new JRadioButton("HTML Report");
        bothRadio = new JRadioButton("Both Formats");

        pdfRadio.setSelected(true);
        RadioButtonsGroup.add(pdfRadio);
        RadioButtonsGroup.add(htmlRadio);
        RadioButtonsGroup.add(bothRadio);

        pdfRadio.setFont(new Font("Arial", Font.PLAIN, 14));
        htmlRadio.setFont(new Font("Arial", Font.PLAIN, 14));
        bothRadio.setFont(new Font("Arial", Font.PLAIN, 14));

        radioButtonsPanel.add(pdfRadio);
```

```
        radioButtonsPanel.add(htmlRadio);
        radioButtonsPanel.add(bothRadio);

        //Add components to main panel
        mainPanel.add(radioButtonsPanel);
        mainPanel.add(Box.createVerticalStrut(10));

        super.IOPanel.add(mainPanel, BorderLayout.CENTER);
        //Set minimum window size
        super.IODialog.setMinimumSize(new Dimension(450, 150));
        //Center the window
        super.IODialog.setLocationRelativeTo(null);
        //Repack window for proper display
        super.IODialog.pack();
    }

    /**
     * Shows the report generation window
     * @throws InputException if there's an error
     */
    @Override
    public void show() throws InputException
    {
        super.IODialog.setLocationRelativeTo(null);
        super.IODialog.setVisible(true);

        String[] reportOptions = getData();

        if (reportOptions != null && reportOptions[0] != null)
        {
            boolean success = generateReport(reportOptions[0]);

            if (success) {
                String successMessage = "Report";
                if (reportOptions[0].equals("both")) {
                    successMessage = "Reports (PDF and HTML)";
                }
                successMessage += " generated";

                super.successOperationWindow(successMessage);
                super.SCSDialog.setVisible(true);
            } else {
                JOptionPane.showMessageDialog(
                    super.IODialog,
                    "Failed to generate report=(",
                    "Error",
                    JOptionPane.ERROR_MESSAGE
                );
            }
        }
```

```
    }

    /**
     * Generates report of selected type
     * @param reportType report type ("pdf", "html" or "both")
     * @return true if generation successful, false otherwise
     */
    private boolean generateReport(String reportType)
    {
        boolean success = false;

        try {
            switch(reportType) {
                case "pdf":
                    success = report.ReportGenerator.generatePDFRe-
port(XML_FILE_PATH, "dog_report.pdf");
                    break;

                case "html":
                    success = report.ReportGenerator.generateHTMLRe-
port(XML_FILE_PATH, "dog_report.html");
                    break;

                case "both":
                    boolean pdfSuccess = report.ReportGenerator.generatePDFRe-
port(XML_FILE_PATH, "dog_report.pdf");
                    boolean htmlSuccess = report.ReportGenerator.generateHTMLRe-
port(XML_FILE_PATH, "dog_report.html");
                    success = pdfSuccess && htmlSuccess;
                    break;

                default:
                    success = false;
            }
        } catch (Exception e) {
            success = false;
        }

        return success;
    }

    /**
     * Gets selected report options
     * @return array with selected options (only report type)
     */
    @Override
    public String[] getData()
    {
        if (super.IOPane.getValue() == null) {
            return null;
```

```
        }

        if (super.IOPane.getValue() instanceof Integer) {
            int option = ((Integer) super.IOPane.getValue()).intValue();

            if (option == JOptionPane.YES_OPTION || option == JOptionPane.OK_OP-
TION)
            {
                String[] results = new String[1];

                //Determine selected report type
                if (pdfRadio.isSelected()) {
                    results[0] = "pdf";
                } else if (htmlRadio.isSelected()) {
                    results[0] = "html";
                } else if (bothRadio.isSelected()) {
                    results[0] = "both";
                } else {
                    results[0] = null;
                }

                return results;
            }
        }

        return null;
    }
}
```

4)DeleteElementWindow.java

```
package ScreenForms;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import list.List;
import object.dog.Dog;


/**
 * Window for deleting rows from the table
 * User can select a row to remove from the data
 * @author Vadim Ustinov
 * @version 1.0
 */
public class DeleteElementWindow extends InputOutputWindow
{
    private JTable deleteTable;
    private List<Dog> dogs;
    private DefaultTableModel tableModel;
    private int rowToDelete;
```

```
    /**
     * Creates window for deleting rows
     * @param table the table to delete rows from
     * @param _dogs the list of dogs to modify
     */
    public DeleteElementWindow(JTable table, List<Dog> _dogs)
    {
        super("Enter the row number to delete", "Delete Row", 1, 0);
        deleteTable = table;
        dogs = _dogs;
        tableModel = (DefaultTableModel) deleteTable.getModel();
    }

    /**
     * Shows the delete window and handles row deletion
     * Asks for row number and removes it after confirmation
     */
    public void show() throws InputException
    {
        showDeleteRow();
    }

    /**
     * Displays the row selection window and handles row number input validation
     * Shows error dialog and retries on invalid row number input
     * Proceeds to data deleting window upon successful row selection
     */
    private void showDeleteRow() throws InputException
    {
        boolean validInput = false;
        boolean operationCancelled = false;
        while (!validInput && !operationCancelled)
        {
            IODialog.setVisible(true);
            String[] rowData = getData();

            if (rowData != null)
            {
                InputException.validRowNumber(rowData[0], tableModel.getRow-
Count());
                rowToDelete = Integer.parseInt(rowData[0]);

                deleteRowByNumber(rowToDelete);
                validInput = true;
            }
            else
            {
                operationCancelled = true;
            }
```

```
        }
    }

    /**
     * Deletes a specific row from the table
     * Asks user to confirm before deleting
     * @param rowToDelete the row to delete (starting from 1)
     */
    private void deleteRowByNumber(int rowToDelete)
    {
        boolean confirmed = confirmOperationWindow("Delete row " + rowToDelete +
"?");

        if (confirmed)
        {
            dogs.remove(rowToDelete-1);
            tableModel.removeRow(rowToDelete-1);

            updateRowNumbers();

            super.successOperationWindow("Row deleted");
        }
    }

    /**
     * Updates the row numbers after deletion
     * Makes sure all rows have correct sequential numbers
     */
    private void updateRowNumbers()
    {
        for (int i = 0; i < tableModel.getRowCount(); i++)
        {
            tableModel.setValueAt(String.valueOf(i+1), i, 0);
        }
    }
}
```

5)EditElementWindow.java

```
package ScreenForms;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import list.List;
import object.dog.Dog;

/**
 * Window for editing existing rows in the table
 * User can select a row and change its data
 * @author Vadim Ustinov
 * @version 2.1
```

```
 */
public class EditElementWindow extends InputOutputWindow
{
    private JTable editTable;
    private DefaultTableModel tableModel;
    private List<Dog> dogs;
    private int rowToEdit;
    private int rowIndex;
    private String[] currentData;
    private boolean operationCancelled;

    private static final String[] FIELD_NAMES = {"Name", "Breed", "Awards"};

    /**
     * Creates window to select which row to edit
     * @param table the table that contains the data to edit
     * @param _dogs the list of dogs to modify
     */
    public EditElementWindow(JTable table, List<Dog> _dogs)
    {
        super("Enter the row number to edit", "Edit Row", 1, 0);
        editTable = table;
        dogs = _dogs;
        tableModel = (DefaultTableModel) editTable.getModel();
    }

    /**
     * Creates window to enter new values for editing with pre-filled current data
     * @param table the table that contains the data to edit
     * @param num_Fields how many fields to edit (name, breed, awards)
     * @param currentData current values of the row being edited
     */
    private EditElementWindow(JTable table, int num_Fields, String[] currentData,
List<Dog> _dogs)
    {
        super("Enter new values: 1 - Name, 2 - Breed, 3 - Awards", "Edit Row", 2,
1);
        editTable = table;
        tableModel = (DefaultTableModel) editTable.getModel();
        dogs = _dogs;

        // Pre-fill text fields with current data
        if (currentData != null && currentData.length >= 3)
        {
            boolean hasAwards=false;
            if(currentData[2] == "1") {
                hasAwards=true;
            }
            textFields[0].setText(currentData[0]); // Name
            textFields[1].setText(currentData[1]); // Breed
```

```
            radioButtons[0].setSelected(hasAwards); // Awards
        }
    }

    /**
     * Main entry point for the editing process
     * Initiates the row selection window to start editing workflow
     */
    public void show() throws InputException
    {
        showRowSelection();
        if (operationCancelled)
        {
            return;
        }
        showDataEditing();
    }

    /**
     * Displays the row selection window and handles row number input validation
     * Shows error dialog and retries on invalid row number input
     * Proceeds to data editing window upon successful row selection
     */
    private void showRowSelection() throws InputException
    {
        boolean validInput = false;
        operationCancelled = false;
        while (!validInput && !operationCancelled)
        {
            IODialog.setVisible(true);
            String[] rowData = getData();

            if (rowData != null)
            {
                InputException.validRowNumber(rowData[0], tableModel.getRow-
Count());
                rowToEdit = Integer.parseInt(rowData[0]);

                currentData = getCurrentRowData(rowToEdit);
                InputException.validDataArray(currentData, 3);

                validInput = true;
            }
            else
            {
                operationCancelled = true;
            }
        }
    }
```

```
    /**
     * Displays the data editing window with pre-filled current values
     * Handles data input validation and table updates
     * Shows error dialog and retries on invalid data input
     */
    private void showDataEditing() throws InputException
    {
        boolean validInput = false;
        operationCancelled = false;
        while (!validInput && !operationCancelled)
        {
            InputException.validRowNumber(String.valueOf(rowToEdit), table-
Model.getRowCount());
            InputException.validDataArray(currentData, 2);

            EditElementWindow editDataWindow = new EditElementWindow(editTable, 2,
currentData, dogs);
            editDataWindow.IODialog.setVisible(true);

            String[] editData = editDataWindow.getData();

            if (editData != null)
            {
                InputException.validEmptyField(editDataWindow.textFields,
FIELD_NAMES);
                InputException.validDataArray(editData, 2);
                InputException.validLettersOnly(editData[0], FIELD_NAMES[0]);
                InputException.validLettersOnly(editData[1], FIELD_NAMES[1]);

                editDataWindow.EditRowByNumber(rowToEdit, editData);

                if (editDataWindow.SCSDialog != null)
                {
                    editDataWindow.SCSDialog.setVisible(true);
                }

                validInput = true;
            }
            else
            {
                operationCancelled = true;
            }
        }
    }

    /**
     * Gets current data from the specified row
     * @param rowToEdit the row number (starting from 1)
     * @return array with current data [name, breed, awards]
     */
```

```
    private String[] getCurrentRowData(int rowToEdit)
    {
        rowIndex = rowToEdit - 1;

        String[] currentData = new String[3];
        currentData[0] = tableModel.getValueAt(rowIndex, 1).toString();
        currentData[1] = tableModel.getValueAt(rowIndex, 2).toString();
        currentData[2] = tableModel.getValueAt(rowIndex, 3).toString();

        return currentData;
    }

    /**
     * Updates the table with edited data for specific row
     * @param rowToEdit the row to edit (starting from 0)
     * @param editedData the new data to set
     */
    private void EditRowByNumber(int rowToEdit, String[] editedData)
    {
        rowIndex = rowToEdit - 1;
        boolean hasAward = false;
        if(editedData[2]=="1")
        {
            hasAward = true;
        }
        Dog newDog = new Dog(editedData[0], editedData[1], hasAward);
        dogs.replace(newDog, rowIndex);
        tableModel.setValueAt(editedData[0].trim(), rowIndex, 1);
        tableModel.setValueAt(editedData[1].trim(), rowIndex, 2);
        tableModel.setValueAt(editedData[2].trim(), rowIndex, 3);

        successOperationWindow("Row edited");
    }
}
```

6)InputException.java

```
package ScreenForms;

import javax.swing.JTextField;

/**
 * Universal exception for handling input errors in the application
 * with validation methods
 * @author Vadim Ustinov
 * @version 1.2
 */
public class InputException extends Exception {
    private final ErrorType;
    private static final long serialVersionUID = 1L;
```

```
    /**
     * Types of errors
     */
    public enum ErrorType {
        EMPTY_FIELD,
        INVALID_NUMBER,
        OUT_OF_RANGE,
        INVALID_FORMAT
    }

    public InputException(String message, ErrorType errorType)
    {
        super(message);
        this.errorType = errorType;
    }

    public ErrorType getErrorType()
    {
        return errorType;
    }

    /**
     * Validates that all required fields are filled
     * @param textFields array of text fields to validate
     * @param fieldNames names of the fields for error messages
     * @throws InputException if any required field is empty
     */
    public static void validEmptyField(JTextField[] textFields, String[] field-
Names) throws InputException
    {
        for (int i = 0; i < textFields.length; i++)
        {
            if (textFields[i].getText().trim().isEmpty())
            {
                String fieldName = (fieldNames != null && i < fieldNames.length) ?
fieldNames[i] : "Field " + (i + 1);
                throw new InputException("Field '" + fieldName + "' cannot be
empty", ErrorType.EMPTY_FIELD);
            }
        }
    }

    /**
     * Validates row number input
     * @param rowNumberStr the row number string to validate
     * @param maxRows maximum allowed row number
     * @throws InputException if row number is invalid or not a number
     */
    public static void validRowNumber(String rowNumberStr, int maxRows) throws In-
putException
```

```
    {
        if (rowNumberStr.trim().isEmpty())
        {
            throw new InputException("Row number cannot be empty", Error-
Type.EMPTY_FIELD);
        }
        try {
            int rowNumber = Integer.parseInt(rowNumberStr);
            if (rowNumber < 1 || rowNumber > maxRows)
            {
                throw new InputException("Invalid row number: " + rowNumber + ".
Valid range: 1-" + maxRows, ErrorType.OUT_OF_RANGE);
            }
        } catch (NumberFormatException e) {
            throw new InputException("Row number must be a number", ErrorType.INVA-
LID_NUMBER);
        }
    }

    /**
     * Validates data array for editing
     * @param data the data array to validate
     * @param expectedLength expected length of the array
     * @throws InputException if data array is invalid
     */
    public static void validDataArray(String[] data, int expectedLength) throws In-
putException
    {
        if (data == null)
        {
            throw new InputException("Data cannot be null",ErrorType.INVALID_FOR-
MAT);
        }
        if (data.length < expectedLength)
        {
            throw new InputException("Invalid data format. Expected " + expected-
Length + " fields, got " + data.length, ErrorType.INVALID_FORMAT);
        }
    }

    /**
     * Validates that text contains only letters and spaces
     * Allows English and Russian letters
     * @param text the text to validate
     * @param fieldName the name of the field for error message
     * @throws InputException if text contains non-letter characters
     */
    public static void validLettersOnly(String text, String fieldName) throws In-
putException
    {
```

```
        if (!text.matches("[a-zA-Za-яA-Я\\s]+"))
        {
            throw new InputException("Field '" + fieldName + "' should contain
only letters", ErrorType.INVALID_FORMAT);
        }
    }

    /**
     * Validates that text contains only numbers 0 or 1
     * Used for binary fields like awards (0 = no awards, 1 = has awards)
     * @param text the text to validate
     * @param fieldName the name of the field for error message
     * @throws InputException if text is not 0 or 1
     */
    public static void validZeroOrOne(String text, String fieldName) throws In-
putException
    {
        if (!text.matches("[01]"))
        {
            throw new InputException("Field '" + fieldName + "' should be only 0
or 1", ErrorType.INVALID_FORMAT);
        }
    }
}
```

7)InputOutputWindow.java

```
package ScreenForms;

import javax.swing.*;

import java.awt.*;


/**
 * Base class for all input and output windows in the application
 * Provides common functionality for dialogs that get user input
 * and show messages to the user
 * @author Vadim Ustinov
 * @version 1.1
 */
abstract public class InputOutputWindow
{
    protected String[] results;
    protected JTextField[] textFields;
    protected JRadioButton[] radioButtons;
    protected ImageIcon icon;
    protected int numFields;
    protected int numRadioButtons;
    protected String title;
    protected JPanel fieldsPanel;
```

```
    protected Font ioDefaultFont;
    protected JPanel IOPanel;
    protected JOptionPane IOPane;
    protected JLabel IOLabel;
    protected JDialog IODialog;
    protected JPanel SCSPanel;
    protected JOptionPane SCSPane;
    protected JLabel SCSLabel;
    protected JDialog SCSDialog;
    protected JPanel CONPanel;
    protected JOptionPane CONPane;
    protected JLabel CONLabel;
    protected JDialog CONDialog;
    /**
     * Makes the application window visible
     * @throws InputException if there's an error displaying the window
     */
    abstract public void show() throws InputException;

    /**
     * Creates a new window for input and output
     * @param text the instruction text to show to user
     * @param title_window the title of the window
     * @param num_Fields how many input fields to create
     * @param _numRadioButtons how many radio buttons to create
     */
    public InputOutputWindow(String text, String title_window, int num_Fields, int
_numRadioButtons)
    {
        // Set up fonts and basic properties
        ioDefaultFont = new Font("Arial", Font.PLAIN, 14);
        numFields = num_Fields;
        numRadioButtons = _numRadioButtons;
        title = title_window;
        results = new String[numFields+numRadioButtons];
        textFields = new JTextField[numFields];
        radioButtons = new JRadioButton[numRadioButtons];
        icon = new ImageIcon("src/picts/dogIcon.png");

        // Create main panel with borders
        IOPanel = new JPanel(new BorderLayout());
        IOPanel.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));

        // Create label with instructions
        IOLabel = new JLabel(text, JLabel.CENTER);
        IOLabel.setFont(ioDefaultFont);

        // Create panel for input fields
        fieldsPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 10, 5));
        // Create each text field and add to panel
```

```
        for (int i = 0; i < numFields; i++)
        {
            textFields[i] = new JTextField(15);
            textFields[i].setFont(ioDefaultFont);
            fieldsPanel.add(textFields[i]);
        }

        for (int i = 0; i < numRadioButtons; i++)
        {
            radioButtons[i] = new JRadioButton("Has the dog any awards?");
            radioButtons[i].setFont(ioDefaultFont);
            fieldsPanel.add(radioButtons[i]);
        }


        // Add label and fields to main panel
        IOPanel.add(IOLabel, BorderLayout.NORTH);
        IOPanel.add(fieldsPanel, BorderLayout.CENTER);

        // Create option pane with Yes/No buttons
        IOPane = new JOptionPane(
            IOPanel,
            JOptionPane.PLAIN_MESSAGE,
            JOptionPane.YES_NO_OPTION
        );

        // Set font for buttons
        UIManager.put("OptionPane.buttonFont", ioDefaultFont);

        // Create the dialog window
        IODialog = IOPane.createDialog(title);
        IODialog.setIconImage(icon.getImage());
    }

    public InputOutputWindow()
    { }


    /**
     * Gets the data that user entered in the text fields
     * @return array of strings with user input, or null if user cancelled
     */
    public String[] getData()
    {
        int currArrSize=0;
        // Check if user clicked Yes/OK button
        if (IOPane.getValue().equals(JOptionPane.YES_OPTION))
        {
            // Get text from each input field
            for (int i = 0; i < numFields; i++)
```

```
            {
                results[currArrSize] = textFields[i].getText();
                currArrSize++;
            }
            for (int i = 0; i < numRadioButtons; i++)
            {
                if(radioButtons[i].isSelected()) {
                    results[currArrSize] = "1";
                }
                else {
                    results[currArrSize] = "0";
                }
                currArrSize++;
            }

            return results;
        }
        return null;
    }

    /**
     * Shows a window with success message
     * @param text the success message to show
     */
    public void successOperationWindow(String text)
    {
        // Create success message
        String message = text + " successfully!";

        // Create panel for message
        SCSPanel = new JPanel(new BorderLayout());
        SCSPanel.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));

        // Create label with message
        SCSLabel = new JLabel(message, JLabel.CENTER);
        SCSLabel.setFont(ioDefaultFont);

        // Create option pane for message
        SCSPane = new JOptionPane(
                SCSLabel,
                JOptionPane.DEFAULT_OPTION
                );

        // Set font for buttons
        UIManager.put("OptionPane.buttonFont", ioDefaultFont);

        // Create dialog window
        SCSDialog = SCSPane.createDialog(title);
        SCSDialog.setIconImage(icon.getImage());
    }
```

```java
/**
 * Shows a window to confirm an action with user
 * @param text the question to ask user
 * @return true if user clicked Yes, false if user clicked No
 */
public boolean confirmOperationWindow(String text)
{
    // Create panel for confirmation message
    CONPanel = new JPanel(new BorderLayout());
    CONPanel.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));

    // Create label with question
    CONLabel = new JLabel(text, JLabel.CENTER);
    CONLabel.setFont(ioDefaultFont);

    // Create option pane with Yes/No buttons
    CONPane = new JOptionPane(
            CONLabel,
            JOptionPane.PLAIN_MESSAGE,
            JOptionPane.YES_NO_OPTION
            );

    // Set font for buttons
    UIManager.put("OptionPane.buttonFont", ioDefaultFont);

    // Create and show dialog window
    CONDialog = CONPane.createDialog(title);
    CONDialog.setIconImage(icon.getImage());
    CONDialog.setVisible(true);

    // Check if user confirmed
    if (CONPane.getValue().equals(JOptionPane.YES_OPTION))
    {
        return true;
    }
    return false;
}
/**
 * Shows an error message dialog
 * @param errorMessage the error message to display
 */
public void showErrorDialog(String errorMessage) {
    JOptionPane.showMessageDialog(
        IODialog,
        errorMessage,
        "Error",
        JOptionPane.ERROR_MESSAGE
    );
}
```

```
}
```

9.3 Пакет report

1)ReportGenerator.java

```
package report;

import net.sf.jasperreports.engine.*;
import net.sf.jasperreports.engine.data.JRMapCollectionDataSource;
import fileManager.FileManager;
import list.List;
import object.dog.Dog;
import java.util.*;

/**
 * Report generation tool
 * Generates reports in PDF and HTML formats from XML data
 */
public class ReportGenerator {

    // Paths to JRXML templates
    private static final String PDF_TEMPLATE = "src/report/pdf_report.jrxml";
    private static final String HTML_TEMPLATE = "src/report/html_report.jrxml";

    /**
     * Generates PDF report from XML data
     * @param xmlPath Path to source XML file with dog data
     * @param outputPath Path where PDF file will be saved
     * @return true if generation successful, false otherwise
     */
    public static boolean generatePDFReport(String xmlPath, String outputPath) {
        try {
            //Load data from XML file
            FileManager = new FileManager();
            List<Dog> dogs = fileManager.inputFromXML(xmlPath);

            if (dogs.getSize() == 0) {
                return false;
            }

            //Convert data to Map format for JasperReports
            java.util.List<java.util.Map<String, Object>> dataList = convertTo-
MapList(dogs);

            //Calculate statistics for report
            int awardsCount = countAwards(dogs);
            double percentage = calculatePercentage(awardsCount, dogs.getSize());

            //Create report parameters (metadata and statistics)
```

```
            java.util.Map<String, Object> params = createParams(xmlPath, dogs.get-
Size(),
                                                                awardsCount, percent-
age);

            //Check if template file exists
            java.io.File templateFile = new java.io.File(PDF_TEMPLATE);
            if (!templateFile.exists()) {
                return false;
            }

            ///Generate PDF report using JasperReports
            //Compile JRXML template
            JasperReport report = JasperCompileManager.compileReport(PDF_TEMPLATE);

            //Create data source from converted data
            JRMapCollectionDataSource dataSource = new JRMapCollectionData-
Source((java.util.Collection) dataList);

            //Fill report with data and parameters
            JasperPrint print = JasperFillManager.fillReport(report, params, data-
Source);

            //Export filled report to PDF file
            JasperExportManager.exportReportToPdfFile(print, outputPath);

            return true;

        } catch (Exception e) {
            return false;
        }
    }

    /**
     * Generates HTML report from XML data
     * @param xmlPath Path to source XML file with dog data
     * @param outputPath Path where HTML file will be saved
     * @return true if generation successful, false otherwise
     */
    public static boolean generateHTMLReport(String xmlPath, String outputPath) {
        try {
            //Load data from XML file
            FileManager = new FileManager();
            List<Dog> dogs = fileManager.inputFromXML(xmlPath);

            if (dogs.getSize() == 0) {
                return false;
            }

            //Convert data to Map format for JasperReports
```

```
            java.util.List<java.util.Map<String, Object>> dataList = convertTo-
MapList(dogs);

            //Calculate statistics for report
            int awardsCount = countAwards(dogs);
            double percentage = calculatePercentage(awardsCount, dogs.getSize());

            //Create report parameters (metadata and statistics)
            java.util.Map<String, Object> params = createParams(xmlPath, dogs.get-
Size(),
                                                                awardsCount, percent-
age);

            //Check if template file exists
            java.io.File templateFile = new java.io.File(HTML_TEMPLATE);
            if (!templateFile.exists()) {
                return false;
            }

            ///Generate HTML report using JasperReports
            //Compile JRXML template
            JasperReport report = JasperCompileManager.compileReport(HTML_TEM-
PLATE);

            //Create data source from converted data
            JRMapCollectionDataSource dataSource = new JRMapCollectionData-
Source((java.util.Collection) dataList);

            //Fill report with data and parameters
            JasperPrint print = JasperFillManager.fillReport(report, params, data-
Source);

            //Export filled report to HTML file
            JasperExportManager.exportReportToHtmlFile(print, outputPath);

            return true;

        } catch (Exception e) {
            return false;
        }
    }


    /**
     * Counts how many dogs have awards
     * @param dogs List of Dog objects to analyze
     * @return Number of dogs with awards
     */
    private static int countAwards(List<Dog> dogs) {
        int count = 0;
```

```
        for (int i = 0; i < dogs.getSize(); i++) {
            if (dogs.at(i).hasAward()) count++;
        }
        return count;
    }

    /**
     * Calculates percentage of dogs with awards
     * @param awardsCount Number of dogs with awards
     * @param total number of dogs
     * @return Percentage value (0.0 to 100.0)
     */
    private static double calculatePercentage(int awardsCount, int total) {
        return total > 0 ? (awardsCount * 100.0 / total) : 0;
    }

    /**
     * Converts List<Dog> to List<Map> format for JasperReports
     * Each Map represents one row in the report with field names matching JRXML
template
     * @param dogs List of Dog objects to convert
     * @return List of Maps with report data
     */
    private static java.util.List<java.util.Map<String, Object>> convertTo-
MapList(List<Dog> dogs) {
        java.util.List<java.util.Map<String, Object>> dataList = new java.util.Ar-
rayList<>();

        for (int i = 0; i < dogs.getSize(); i++) {
            Dog = dogs.at(i);
            java.util.Map<String, Object> row = new java.util.HashMap<>();

            // Map keys must match field names in JRXML templates
            row.put("id", i + 1);
            row.put("name", dog.getName());
            row.put("breed", dog.getBreed());
            row.put("awardsText", dog.hasAward() ? "YES" : "NO");
            row.put("awards", dog.hasAward());

            dataList.add(row);
        }

        return dataList;
    }

    /**
     * Creates parameters for JasperReports templates
     * These parameters are used in report headers, footers, and summary sections
     * @param xmlPath Source XML file path
     * @param totalDogs Total number of dogs
```

```
     * @param awardsCount Number of dogs with awards
     * @param percentage of dogs with awards
     * @return Map of report parameters
     */
    private static java.util.Map<String, Object> createParams(String xmlPath, int
totalDogs,
                                                              int awardsCount, dou-
ble percentage) {
        java.util.Map<String, Object> params = new java.util.HashMap<>();

        // Report metadata
        params.put("REPORT_TITLE", "Dog Festival Report");
        params.put("LAB_NUMBER", "Laboratory Work #7");
        params.put("GENERATION_DATE", new Date());
        params.put("SOURCE_FILE", xmlPath);

        // Report statistics
        params.put("TOTAL_DOGS", totalDogs);
        params.put("AWARDS_COUNT", awardsCount);
        params.put("AWARDS_PERCENTAGE", percentage);

        return params;
    }
}
```

9.3 Пакет list

1)List.java

```
package list;

/**
 * Class for working with a list of items
 * @param <T> type of list items
 * @author Vadim Ustinov
 * @version 1.0
 */
public class List<T> implements interfaceList<T>
{
    private static class Node<T>
    {
        T val;
        Node<T> next;
        Node(T data)
        {
            this.val = data;
            this.next = null;
        }
    }

    private Node<T> head;
```

```
    private Node<T> end;
    private int sz;

    public List()
    {
        head = null;
        end = null;
        sz=0;
    }

    /**
     * Checks if the index is valid for access operations
     * @param index to check
     * @throws IndexOutOfBoundsException if index is out of range
     */
    private void checkIndex(int index)
    {
        if(index<0 || index>=sz)
        {
            throw new IndexOutOfBoundsException("Index " + index + " is out of
range (list size: " + sz + ")\n");
        }
    }

    /**
     * Checks if the index is valid for insert operations
     * @param index to check
     * @throws IndexOutOfBoundsException if index is out of range
     */
    private void checkIndexForInsert(int index)
    {
        if(index<0 || index>sz)
        {
            throw new IndexOutOfBoundsException("Index " + index + " is out of
range (list size: " + sz + ")\n");
        }
    }

    /**
     * Adds an element to the end of the list
     * @param value to add
     */
    @Override
    public void push_back(T value)
    {
        Node<T> newNode = new Node<>(value);
        if(head==null)
        {
            head = newNode;
            end = newNode;
```

```
            head.next = null;
            end.next = null;
        }
        else
        {
            end.next = newNode;
            end = newNode;
        }
        sz++;
    }

    /**
     * Adds an element to the beginning of the list
     * @param value to add
     */
    @Override
    public void push_front(T value)
    {
        Node<T> newNode = new Node<>(value);
        if(head == null)
        {
            head = newNode;
            end = newNode;
            head.next = null;
            end.next = null;
        }
        else
        {
            Node<T> temp = head;
            head = newNode;
            head.next = temp;
        }
        sz++;
    }

    /**
     * Inserts an element at the specified position
     * @param value to insert
     * @param index insertion position
     */
    @Override
    public void insert(T value, int index)
    {
        checkIndexForInsert(index);
        if(index==0)
        {
            this.push_front(value);
            return;
        }
        if(index==sz)
```

```
        {
            this.push_back(value);
            return;
        }
        Node<T> newNode = new Node<>(value);
        Node<T> curr = head;
        for(int i=0;i<index-1;i++)
        {
            curr=curr.next;
        }
        newNode.next = curr.next;
        curr.next = newNode;
        sz++;
    }

    /**
     * Returns the number of elements in the list
     * @return size of the list
     */
    @Override
    public int getSize() {return sz;}

    /**
     * Returns the element at the specified position
     * @param index position of the element
     * @return element at the specified position
     */
    @Override
    public T at(int index)
    {
        checkIndex(index);
        Node<T> curr = head;
        for(int i=0;i<index;i++)
        {
            curr = curr.next;
        }
        return curr.val;
    }

    /**
     * Removes the element at the specified position
     * @param index position of the element to remove
     */
    @Override
    public void remove(int index)
    {
        checkIndex(index);
        if(index==0)
        {
            head = head.next;
```

```
            if(head==null)
            {
                end=null;
            }
            sz--;
            return;
        }
        Node<T> curr = head, temp;
        for(int i=0;i<index-1;i++)
        {
            curr=curr.next;
        }
        temp = curr.next;
        curr.next = temp.next;
        if(curr.next == null)
        {
            end = curr;
        }
        temp = null;
        sz--;
    }

    /**
     * Checks if the list is empty
     * @return true if the list is empty, false otherwise
     */
    @Override
    public boolean isEmpty()
    {
        if(head!=null)
        {
            return false;
        }
        return true;
    }

    /**
     * Checks if the list contains the specified value
     * @param value to search for
     * @return true if the value is found, false otherwise
     */
    @Override
    public boolean contains(T value)
    {
        Node<T> curr = head;
        while(curr!=null)
        {
            if(curr.val==value)
            {
                return true;
```

```
            }
            curr=curr.next;
        }
        return false;
    }

    /**
     * Removes all elements from the list
     */
    @Override
    public void clear()
    {
        sz=0;
        head=null;
        end=null;
    }

    /**
     * Replaces the element at the specified position with a new value
     * @param value new value
     * @param index position of the element to replace
     */
    @Override
    public void replace(T value, int index)
    {
        checkIndex(index);
        Node<T> curr = head;
        for(int i=0;i<index;i++)
        {
            curr = curr.next;
        }
        curr.val = value;
    }

    /**
     * Converts the list to an array of strings
     * @return array of string representations of elements, or null if list is
empty
     */
    @Override
    public String[] convToStr()
    {
        if(sz==0)
        {
            return null;
        }
        String[] res = new String[sz];
        Node<T> curr = head;
        for(int i=0;i<sz;i++)
        {
```

```
            res[i] = curr.val.toString();
            curr = curr.next;
        }
        return res;
    }
}
```

9.4 Пакет laba

1)DogList.java

```
import ScreenForms.InputException;
import ScreenForms.MainWindow;

/**
 * Main function
 * @author Vadim Ustinov
 * @version 1.0
 */
public class DogList {
    public static void main(String[] args) throws InputException
    {
        MainWindow main = new MainWindow();
        main.show();
    }
}
```