Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Лабораторная работа №5

«Разработка диаграммы классов»

Студент: Беласин Д.А.

ФИТ 3 курс 1 группа

Преподаватель: Якубенко К.Д.

Минск 2024

1. **Постановка задачи**

Задачей данной работы является детальное описание структуры взаимодействия классов, используемых в web-приложении, на основе данных, описанных в лабораторной работе №1. Это предполагает построение точной и понятной модели, которая поможет глубже понять, как классы приложения взаимодействуют друг с другом, какие они выполняют роли, какие атрибуты и методы содержат, а также какие связи существуют между ними.

Описание назначения диаграммы классов:

Диаграмма классов — это важнейший элемент UML, позволяющий проектировщикам, разработчикам и другим участникам процесса разработки наглядно представлять архитектуру системы. Диаграмма классов играет ключевую роль в разработке приложений, так как она демонстрирует:

* Структуру приложения — как организованы его сущности.
* Взаимодействие классов — какие классы связаны между собой и каким образом (ассоциации, наследование, композиция и другие типы связей).
* Функции и данные — какие атрибуты и методы принадлежат каждому классу.

Цель данной работы заключается в разработке диаграммы классов для платформы аренды парковочных мест, которая будет служить основой для последующего создания кода. Это поможет:

* Проанализировать функциональность системы.
* Упростить коммуникацию между разработчиками.
* Обеспечить базу для дальнейших расширений и доработок приложения.

Задачей данной работы является построение диаграммы классов для интерфейса аренды парковочных мест, что включает в себя определение классов, их атрибутов, методов и связей между ними. Разработка такой диаграммы помогает визуализировать логику приложения, обеспечить правильное взаимодействие компонентов и подготовить основу для группировки этих классов в пакеты для лучшей структурированности.

Также необходимо составить диаграмму пакетов системы. Они облегчают общение между участниками проекта (разработчиками, тестировщиками, заказчиками), так как даёт наглядное представление о том, как система организована, а также диаграмма классов позволяет лучше понимать существующий код и упрощает его расширение.

# 2. Описание программных средств

Описание программного средства Draw.io:

* Название: Draw.io.
* Версия: Актуальная версия (по состоянию на 2024 год). Программа обновляется регулярно.
* Разработчик: JGraph Ltd.
* Адрес загрузки: https://app.diagrams.net/. Draw.io доступен как онлайн-приложение, так и в виде настольного клиента для различных операционных систем.
* Режим использования: Draw.io предоставляет бесплатный доступ ко всем своим функциям без ограничений. Это полностью бесплатный инструмент с открытым исходным кодом, доступный как для личного, так и для корпоративного использования.
* Платформы: Draw.io доступен как веб-приложение, которое работает в любом современном браузере. Также доступны настольные версии для Windows, macOS и Linux.
* Типы моделей, с которыми работает Draw.io: Draw.io используется для создания диаграмм, схем, моделей процессов и прочих визуальных структур. Программа поддерживает работу с блок-схемами, диаграммами UML, IDEF0, сетевыми диаграммами, архитектурными схемами, организационными диаграммами и многими другими визуальными представлениями данных.
* Основные функции: Создание и редактирование диаграмм различных типов (например, UML, блок-схемы, BPMN, ERD). Поддержка импорта и экспорта файлов в различных форматах, включая XML, PNG, SVG и PDF. Возможность совместной работы с другими пользователями через облачные сервисы (Google Drive, OneDrive, Dropbox и др.). Функции автосохранения и интеграции с популярными системами управления версиями. Поддержка версии оффлайн, которая позволяет работать без подключения к интернету. Гибкая система настройки шаблонов и библиотек элементов для ускорения работы.

Draw.io — это удобный, бесплатный инструмент для создания диаграмм и схем, который поддерживает совместную работу и интеграцию с облачными сервисами, обеспечивая гибкость и доступность на различных платформах.

**3.** **Практическое задание**

Диаграмма классов для нашего проекта будет выглядеть следующим образом – Рисунок 3.1.

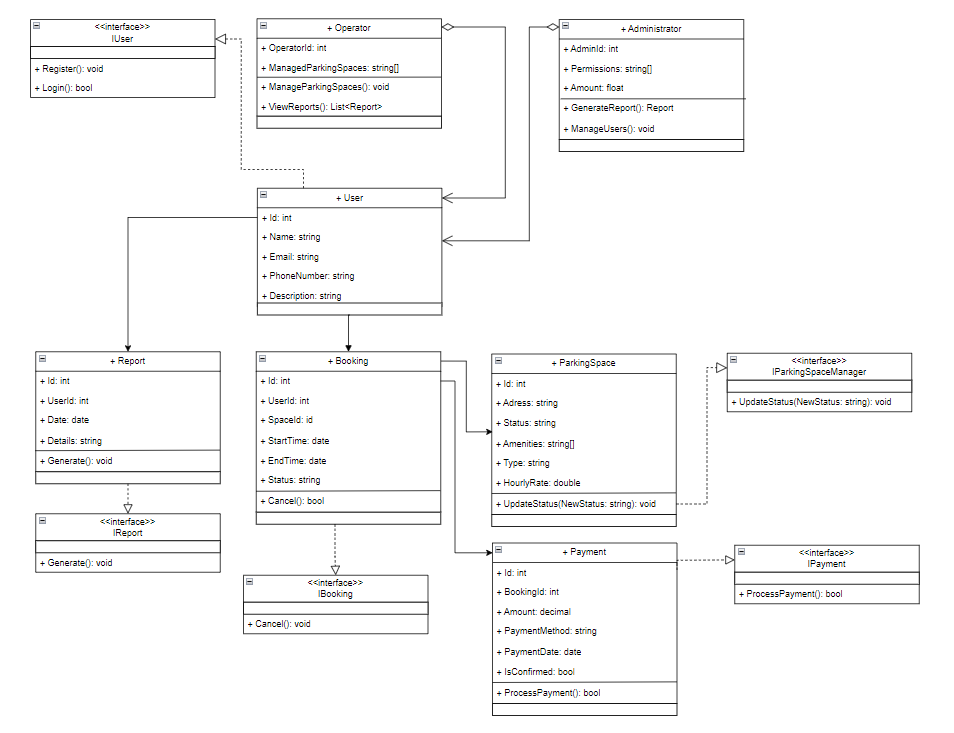


Рисунок 3.1 – Диаграмма классов

Эти классы представляют ключевые сущности, отвечающие за хранение данных и предоставление функциональности через методы. Каждый класс включает набор атрибутов и методов, обеспечивающих их базовое поведение.

Кроме того, в системе используются интерфейсы:

* IUser, который определяет общие действия для всех пользователей.
* IReport, описывающий возможности отправки репорта.
* IBooking, предоставляющий функционал для бронирования.
* IPayment,предоставляющий функционал для оплаты.
* IParkingSpaceManager,для управления парковочными местами

Связи между классами:

User:

* Связь с Booking. Один пользователь может иметь несколько бронирований, но каждое бронирование может существовать независимо от пользователя.
* Связь с Parking. Пользователь может взаимодействовать с несколькими парковочными местами, но это не требует жесткой зависимости.

Parking:

* Связь с Booking. Парковочное место может быть забронировано, но не зависит от бронирования. При удалении бронирования парковочное место остается доступным.
* Связь с User. Указывает на то, что пользователь может просматривать и выбирать доступные парковочные места.

Booking:

* Связь с User и Parking. Бронирование связано с пользователем и парковочным местом, но эти объекты могут существовать независимо.
* Связь с Payment. Бронирование может иметь связанные платежи, но они не являются обязательными для существования бронирования.

Payment:

* Связь с Booking. Платежи зависят от бронирования, но могут быть обработаны независимо.

Структура классов:

User:

* Хранит информацию о пользователе, его предпочтениях и связанных бронированиях.

Parking:

* Содержит данные о парковочных местах, такие как местоположение, доступность и тип.

Booking:

* Управляет процессом бронирования парковочных мест, включая информацию о сроках и связанных пользователях.

Payment:

* Обрабатывает информацию о платежах, связанных с бронированиями.

Заключение:

Такое распределение обязанностей между классами и четкие связи между ними позволяют упростить реализацию и расширение функциональности системы аренды парковочных мест. Каждому классу назначены свои обязанности, что облегчает поддержку и модификацию системы.

Далее рассмотрим диаграмму компонентов на рисунке 3.2:

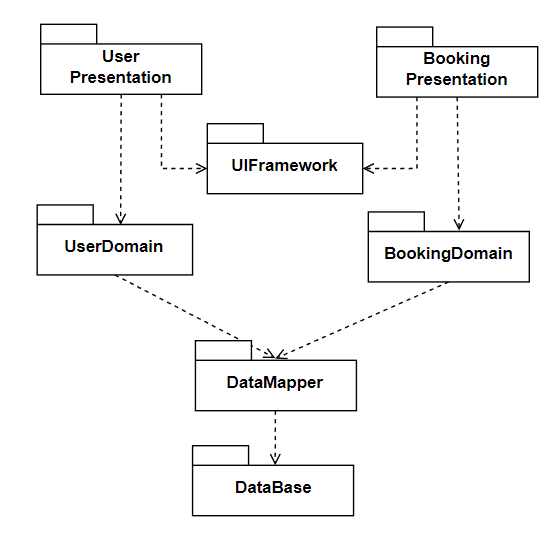


Рисунок 3.2 – Диаграмма пакетов

Связи между пакетами:

* UserPresentation взаимодействует с UserDomain для отображения данных.
* BookingPresentation использует BookingDomain для работы с данными о парковочных местах.
* UserDomain и BookingDomain связаны с DataMapper, который преобразует их данные для Database.
* UIFramework связывает визуальные и логические компоненты, предоставляя общий интерфейс.

Эта структура обеспечивает модульность и четкое разделение обязанностей между пакетами, что делает приложение легко поддерживаемым и расширяемым.

# Теоретические вопросы подготовки к лабораторной работе

1. Перечислите структурные диаграммы, которые входят в UML 2.0:

В UML 2.0 включено несколько структурных диаграмм, каждая из которых фокусируется на описании статической архитектуры системы:

* Диаграмма классов (Class Diagram) — описывает классы, их атрибуты, методы и связи между классами.
* Диаграмма объектов (Object Diagram) — показывает конкретные экземпляры классов и их связи в определённый момент времени.
* Диаграмма пакетов (Package Diagram) — группирует элементы системы (классы, компоненты) в пакеты для организации архитектуры.
* Диаграмма компонентов (Component Diagram) — описывает физические модули системы и взаимодействие между ними.
* Диаграмма развёртывания (Deployment Diagram) — демонстрирует распределение компонентов системы по узлам физической инфраструктуры.
* Диаграмма композитной структуры (Composite Structure Diagram) — описывает внутреннюю структуру компонентов и классов и их взаимосвязи на более детальном уровне.
* Диаграмма профилей (Profile Diagram) — используется для создания расширений UML и применяет пользовательские стереотипы и другие механизмы для конкретных областей.

2. Укажите назначение структурных диаграмм:

Структурные диаграммы в UML предназначены для моделирования статической архитектуры системы. Они фокусируются на описании:

* элементов системы (классов, объектов, компонентов, узлов и пакетов);
* иерархии и связей между элементами;
* взаимодействий и зависимостей между различными компонентами системы на уровне классов и физических модулей.

Эти диаграммы помогают разработчикам и архитекторам чётко понимать структуру системы и её составляющих, описывать модели данных и взаимосвязи между объектами, а также разрабатывать архитектуру программного обеспечения с учётом требований к физической и логической структуре системы.

3. Опишите нотации, которые используются для построения Classes диаграмм:

Для построения диаграмм классов (Class Diagram) используются нотации класса, связей между классами, расширения, стереотип. Класс — представляется прямоугольником, который разделён на три части: верхняя часть — имя класса, средняя часть — атрибуты (свойства) класса, нижняя часть — методы (операции) класса.

Связи между классами:

* Ассоциация — линия между классами, которая обозначает, что один класс взаимодействует с другим. Может быть направленной (стрелка) или недиректной.
* Агрегация — полая ромбовидная стрелка, указывающая на «целое» и «части» (например, класс «Автомобиль» состоит из класса «Колесо»).
* Композиция — закрашенная ромбовидная стрелка, показывающая, что «часть» не может существовать без «целого» (например, «Комната» является частью «Дома»).
* Наследование (генерализация) — линия с треугольной стрелкой, указывающая на родительский класс, от которого наследуются дочерние классы.
* Зависимость — пунктирная линия со стрелкой, обозначающая, что один класс зависит от другого (например, использование одного класса в качестве параметра метода другого класса).

Интерфейс — изображается как прямоугольник с надписью <<interface>> или как круг (класс реализует интерфейс). Абстрактные классы и методы — обозначаются курсивом.

4. Для чего применяются расширения диаграмм UML:

Расширения UML применяются для того, чтобы адаптировать или настраивать язык UML под специфические потребности проекта или предметной области. Стандарт UML предоставляет гибкие механизмы для создания собственных обозначений и терминов, которые могут расширять его базовые элементы.

5. Что означают понятия «стереотип» и «тегированное значение» в контексте расширенных диаграмм:

Стереотип — это механизм расширения UML, который используется для добавления дополнительной информации к существующим элементам диаграммы. Например, можно создать стереотип <<controller>> в MVC-архитектуре.

Тегированное значение (tagged value) — это дополнительный атрибут или метка, которая присваивается элементу диаграммы для уточнения его характеристик. Тегированное значение дополняет информацию, которую нельзя выразить стандартными средствами UML.