Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный технологический университет»

Кафедра информационных систем и технологий

**Отчет по лабораторной работе на тему**:

**«Объектно-ориентированное моделирование. Структурные диаграммы UML»**

Выполнил:

студент 4 курса 7 группы ФИТ

Леонов Д.И.

Цель:

Изучение методологии объектно-ориентированного моделирования средствами UML. Ознакомление с основными принципами объектно-ориентированного проектирования программного обеспечения, получение навыков проектирования структуры информационной системы с применением UML.

Минск 2023

# 1. Теоретические вопросы

## 1.1 Перечислите структурные диаграммы, которые входят в UML 2.0.

Структурные UML диаграммы:

- Диаграмма классов;

- Диаграмма компонентов;

- Диаграмма объектов;

- Диаграмма развёртывания;

- Диаграмма пакетов;

- Диаграмма профиля;

- Составная структурная диаграмма;

## 1.2 Укажите назначение структурных диаграмм.

Структурные диаграммы представляют статическую структуру программного обеспечения или системы, они также показывают различные уровни абстракции и реализации. Они используются, чтобы помочь визуализировать различные структуры, составляющие систему, например, базу данных или приложение. Они показывают иерархию компонентов или модулей и то, как они связаны и взаимодействуют между собой. Эти инструменты обеспечивают руководство работы и гарантируют, что все части системы функционируют так, как задумано по отношению ко всем остальным частям.

## 1.3 Опишите нотации, которые используются для построения Classes диаграмм.

UML включает три вида строительных блоков:

1) диаграммы;

2) сущности:

- Структурные:

Это «имена существительные» в модели UML. В основном, статические части модели, представляющие либо концептуальные, либо физические элементы. Основным видом структурной сущности в диаграммах классов является класс.

- Поведенческие:

Динамические части моделей UML. Это «глаголы» моделей, представляющие поведение модели во времени и пространстве. Основной из них является взаимодействие – поведение, которое заключается в обмене сообщениями между наборами объектов или ролей в определенном контексте для достижения некоторой цели. Сообщение изображается в виде линии со стрелкой, почти всегда сопровождаемой именем операции.

- Аннотирующие:

Это поясняющие части UML-моделей, иными словами, комментарии, которые можно применить для описания, выделения и пояснения любого элемента модели. Главная из аннотирующих сущностей – примечание. Это символ, служащий для описания ограничений и комментариев, относящихся к элементу либо набору элементов. Графически представлен прямоугольником с загнутым углом; внутри помещается текстовый или графический комментарий.

3) Связи.

К структурным сущностям относится:

1) Класс – это описание набора объектов с одинаковыми атрибутами, операциями, связями и семантикой.

Графически класс изображается в виде прямоугольника, разделенного на 3 блока горизонтальными линиями:

- имя класса;

- атрибуты (свойства) класса;

- операции (методы) класса.

Для атрибутов и операций может быть указан один из трех типов видимости:

* (-) - private (частный);
* # - protected (защищенный);
* + - public (общий);

Видимость для полей и методов указывается в виде левого символа в строке с именем соответствующего элемента.

Каждый класс должен обладать именем, отличающим его от других классов. Имя – это текстовая строка. Имя класса может состоять из любого числа букв, цифр и знаков препинания (за исключением двоеточия и точки) и может записываться в несколько строк.

На практике обычно используются краткие имена классов, взятые из словаря моделируемой системы. Каждое слово в имени класса традиционно пишут с заглавной буквы (верблюжья конвенция), например Sensor (Датчик) или TemperatureSensor (ДатчикТемпературы).

2) Атрибут (свойство) – это именованное свойство класса, описывающее диапазон значений, которые может принимать экземпляр атрибута. Класс может иметь любое число атрибутов или не иметь ни одного. В последнем случае блок атрибутов оставляют пустым.

Атрибут представляет некоторое свойство моделируемой сущности, которым обладают все объекты данного класса. Имя атрибута, как и имя класса, может представлять собой текст. На практике для именования атрибута используются одно или несколько коротких существительных, выражающих некое свойство класса, к которому относится атрибут.

Можно уточнить спецификацию атрибута, указав его тип, кратность (если атрибут представляет собой массив некоторых значений) и начальное значение по умолчанию. Статические атрибуты класса обозначаются подчеркиванием.

3) Операция (метод) – это реализация метода класса. Класс может иметь любое число операций либо не иметь ни одной. Часто вызов операции объекта изменяет его атрибуты. Графически операции представлены в нижнем блоке описания класса.

Допускается указание только имен операций. Имя операции, как и имя класса, должно представлять собой текст. На практике для именования операции используются короткие глагольные конструкции, описывающие некое поведение класса, которому принадлежит операция. Обычно каждое слово в имени операции пишется с заглавной буквы, за исключением первого, например move (переместить) или isEmpty (проверка на пустоту).

Можно специфицировать операцию, устанавливая ее сигнатуру, включающую имя, тип и значение по умолчанию всех параметров, а применительно к функциям – тип возвращаемого значения.

Абстрактные методы класса обозначаются курсивным шрифтом. Статические методы класса обозначаются подчеркиванием.

Изображая класс, не обязательно показывать сразу все его атрибуты и операции. Для конкретного представления, как правило, существенна только часть атрибутов и операций класса. В силу этих причин допускается упрощенное представление класса, то есть для графического представления выбираются только некоторые из его атрибутов. Если помимо указанных существуют другие атрибуты и операции, вы даете это понять, завершая каждый список многоточием.

Чтобы легче воспринимать длинные списки атрибутов и операций, желательно снабдить префиксом (именем стереотипа) каждую категорию в них. В данном случае стереотип – это слово, заключенное в угловые кавычки, которое указывает то, что за ним следует.

### 1.3.1 Типы связей

Существует четыре типа связей в UML:

- Зависимость

- Ассоциация

- Обобщение

- Реализация

Эти связи представляют собой базовые строительные блоки для описания отношений в UML, используемые для разработки хорошо согласованных моделей.

Первая из них – зависимость – семантически представляет собой связь между двумя элементами модели, в которой изменение одного элемента (независимого) может привести к изменению семантики другого элемента (зависимого). Графически представлена пунктирной линией, иногда со стрелкой, направленной к той сущности, от которой зависит еще одна; может быть снабжена меткой.

Зависимость

Рисунок 1.1 – Зависимость

Зависимость – это связь использования, указывающая, что изменение спецификаций одной сущности может повлиять на другие сущности, которые используют ее.

Ассоциация – это структурная связь между элементами модели, которая описывает набор связей, существующих между объектами.

Ассоциация показывает, что объекты одной сущности (класса) связаны с объектами другой сущности таким образом, что можно перемещаться от объектов одного класса к другому.

Например, класс Человек и класс Школа имеют ассоциацию, так как человек может учиться в школе. Ассоциации можно присвоить имя «учится в». В представлении однонаправленной ассоциации добавляется стрелка, указывающая на направление ассоциации.

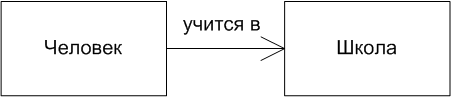


Рисунок 1.2 – Ассоциация

Двойные ассоциации представляются линией без стрелок на концах, соединяющей два классовых блока.

Ассоциация может быть именованной, и тогда на концах представляющей её линии будут подписаны роли, принадлежности, индикаторы, мультипликаторы, видимости или другие свойства.

Множественность ассоциации представляет собой диапазон целых чисел, указывающий возможное количество связанных объектов. Он записывается в виде выражения с минимальным и максимальным значением; для их разделения используются две точки. Устанавливая множественность дальнего конца ассоциации, вы указываете, сколько объектов может существовать на дальнем конце ассоциации для каждого объекта класса, находящегося на ближнем ее конце. Количество объектов должно находиться в пределах заданного диапазона. Множественность может быть определена как единица 1, ноль или один 0..1, любое значение 0..\* или \*, один или несколько 1..\*. Можно также задавать диапазон целых значений, например 2..5, или устанавливать точное число, например 3.



Рисунок 1.3. – Множественность ассоциаций

Агрегация – особая разновидность ассоциации, представляющая структурную связь целого с его частями. Как тип ассоциации, агрегация может быть именованной. Одно отношение агрегации не может включать более двух классов (контейнер и содержимое).

Агрегация встречается, когда один класс является коллекцией или контейнером других. Причём, по умолчанию агрегацией называют агрегацию по ссылке, то есть когда время существования содержащихся классов не зависит от времени существования содержащего их класса. Если контейнер будет уничтожен, то его содержимое — нет.

Графически агрегация представляется пустым ромбом на блоке класса «целое», и линией, идущей от этого ромба к классу «часть».

Агрегация

Рисунок 1.4 – Агрегация

Композиция — более строгий вариант агрегации. Известна также как агрегация по значению.

Композиция – это форма агрегации с четко выраженными отношениями владения и совпадением времени жизни частей и целого. Композиция имеет жёсткую зависимость времени существования экземпляров класса контейнера и экземпляров содержащихся классов. Если контейнер будет уничтожен, то всё его содержимое будет также уничтожено.

Графически представляется как и агрегация, но с закрашенным ромбиком.

Композиция

Рисунок 1.5 – Композиция

Обобщение – выражает специализацию или наследование, в котором специализированный элемент (потомок) строится по спецификациям обобщенного элемента (родителя). Потомок разделяет структуру и поведение родителя. Графически обобщение представлено в виде сплошной линии с пустой стрелкой, указывающей на родителя.

Обобщение

Рисунок 1.6 – Обобщение

Четвертая – реализация – это семантическая связь между классами, когда один из них (поставщик) определяет соглашение, которого второй (клиент) обязан придерживаться. Это связи между интерфейсами и классами, которые реализуют эти интерфейсы. Это, своего рода, отношение «целое-часть». Поставщик, как правило, представлен абстрактным классом. В графическом исполнении связь реализации – это гибрид связей обобщения и зависимости: треугольник указывает на поставщика, а второй конец пунктирной линии – на клиента.

Реализация

Рисунок 1.7 – Реализация

## 1.4 Для чего применяются расширения диаграмм UML?

Механизмы расширения UML позволяют отобразить особенности моделей предметных областей. То есть они позволяют модифицировать язык UML в соответствии с потребностями проекта или особенностями технологии разработки.

## 1.5 Что означают понятия «стереотип» и «тегированное значение» в контексте расширенных диаграмм?

Стереотипы (stereotype) - расширяют словарь UML, позволяя на основе существующих элементов языка создавать новые, ориентированные для решения конкретной проблемы; помеченные (тегированные) значения (tagged value) - расширяют свойства основных конструкций UML, позволяя включать дополнительную информацию в спецификацию элемента.

# 2. Техническое задание к проекту «Интерфейс бронирования автомобилей»

Это приложение, позволяющее пользователям любой возрастной группы брать автомобили в аренду.

## 2.1 Функциональные требования

Возможности приложения:

- Пользователь:

* Регистрация/Авторизация;
* Просмотр тарифов;
* Каталог автомобилей;
* Просмотр карты с расположением автомобилей;
* Просмотр информации о профиле, о предыдущих заказах;
* Бронирование авто;
* Пополнение баланса;

- Администратор:

* Авторизация;
* Просмотр информации о пользователях;
* Мониторинг арендованных автомобилей;
* Проверка фотоконтроля автомобиля;
* Тех-поддержка пользователей;

# 3. Описание программных средств

Для создания диаграмм компонентов и развертывания использовалось приложение draw.io. draw.io — инструмент для создания диаграмм, блок-схем, интеллект-карт, бизнес-макетов, отношений сущностей, программных блоков и другого. Сервис распространяется на бесплатной основе с открытым исходным кодом. Draw.io обладает богатым набором функций для визуализации большинства задач пользователя.

Изучение методологии объектно-ориентированного моделирования средствами UML. Ознакомление с основными принципами объектно-ориентированного проектирования программного обеспечения, получение навыков проектирования функциональности информационной системы с применением UML.

Особенности draw.io:

* Более 500 шаблонов элементов и фигур;
* Облегчённый интерфейс, в котором за короткий промежуток времени можно создать готовый проект;
* Поддержка горячих клавиш, задействованных в большинстве графических редакторов;
* Экспорт в форматы: JPG, PNG, SVG, VDSX;
* Возможность совместной работы;
* Наличие различных фоновых тем;
* Мультиязычный интерфейс.

# 4. Описание практического задания

В данной лабораторной работе было необходимо построить диаграмму

классов. На рисунке 4.1 представлена диаграмма с классами уровня модели приложения.

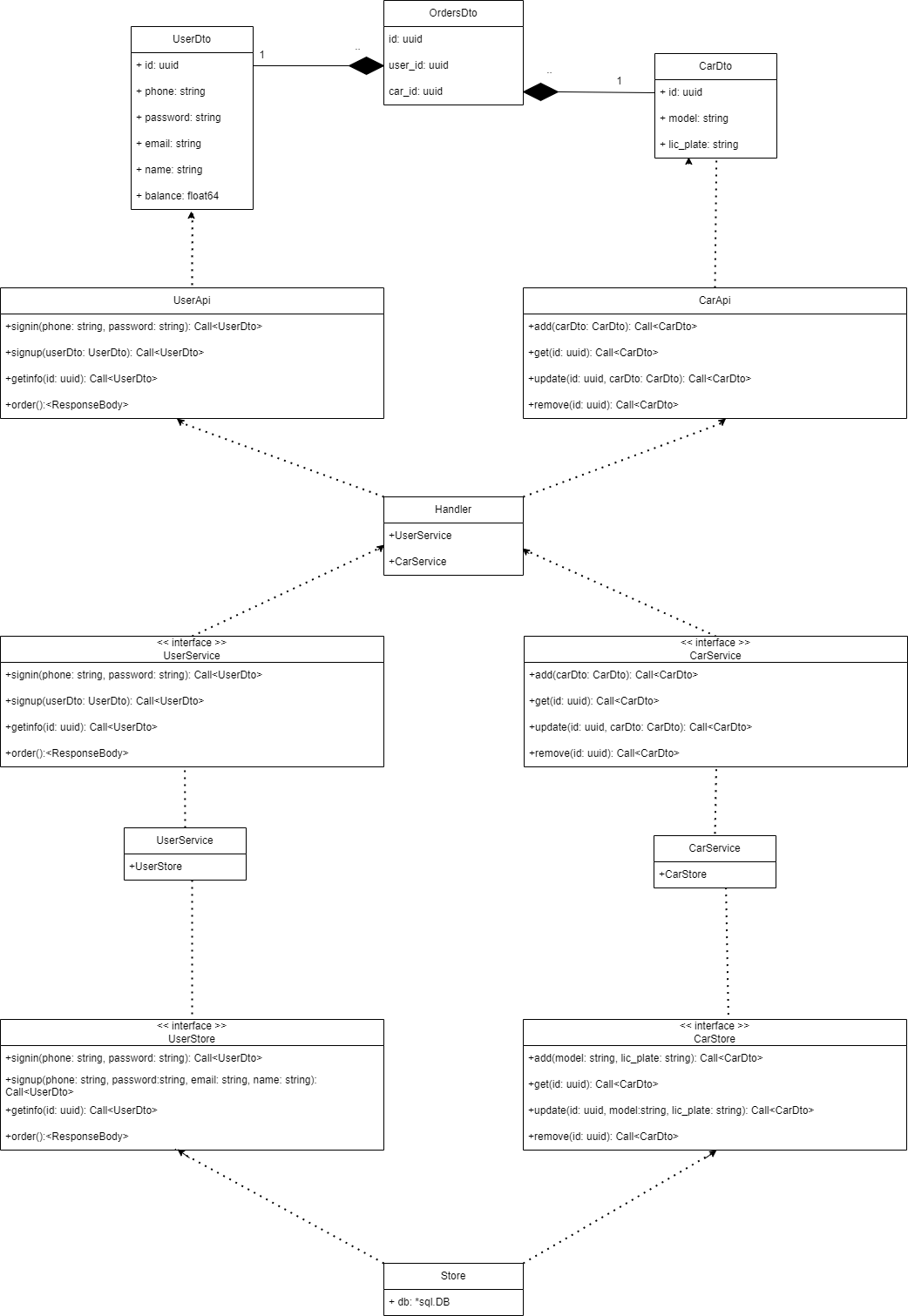


Рисунок 4.1. Классы модели клиентского приложения

В блоках мы указываем название класса, поля и методы, из которых он состоит. Через двоеточие мы обозначаем тип данных поля. Знаком «+» мы говорим о том, что поле доступно для использования вне класса. «1..\*» обозначает тип связи один-ко-многим.

**UserDto**: Структура описывающая пользователей в системе, содержит: уникальный идентификатор пользователя, телефон, пароль, почту, имя и баланс.

**CarDto**: Структура описывающая автомобили в системе, содержит: уникальный идентификатор автомобиля, модель и номерной знак.

**OrderDto**: Структура хранящая информацию о поездках, содержит: уникальный идентификатор поездки, идентификатор пользователя и идентификатор автомобиля.

**UserApi**: Методы определенные для структуры Handler. Они позволяют выставить ендпоинты для работы наружу приложения, чтобы клиенты могли к ним обращаться. Это верхний уровень модели «Чистой архитектуры» в Go. В нем определены методы: регистрации, входа, заказа на бронирование автомобиля, а также получение информации о пользователе.

**CarApi**: Методы определенные для структуры Handler. Они позволяют выставить ендпоинты для работы наружу приложения, чтобы клиенты (в данном случае администратор) могли к ним обращаться. В нем определены следующие методы: добавить автомобиль, удалить автомобиль, изменить информацию об автомобиле, получить информация об автомобиле.

Handler: Структура содержащая в себе интерфейсы для работы с уровнем сервиса.

**UserService <<interface>>**: Методы определенные для структуры UserService. На уровне сервиса происходит вся бизнес-логика: действия с данными, валидация, логирование и так далее. Зачастую сигнатуры методов не отличаются от методов в Handler.

**CarService** **<<interface>>**: Методы определенные для структуры CarService. На уровне сервиса происходит вся бизнес-логика: действия с данными, валидация, логирование и так далее. Зачастую сигнатуры методов не отличаются от методов в Handler.

**UserService**: Структура содержащая в себе интерфейс для работы с уровнем базы данных и определяющая методы взаимодействия с пользователем.

**CarService**: Структура содержащая в себе интерфейс для работы с уровнем базы данных и определяющая методы взаимодействия с автомобилями.

**UserStore <<interface>>**: Методы определенные для структуры Store. Эти методы описывают работу с пользователем на уровне базы данных. Там не происходит никаких действий, кроме работы с базой данных.

**CarStore <<interface>>**: Методы определенные для структуры Store. Эти методы описывают работу с автомобилями на уровне базы данных. Там не происходит никаких действий, кроме работы с базой данных.

**Store**: Структура содержащая все необходимое для работы с базой данных. В данном случае это переменная db c типом \*sql.DB.

Описание методов UserApi:

* +signin(phone: string, password: string): Call<UserDto> - метод для входа в систему, принимающий номер телефона и пароль и возвращающий информацию о пользователе.
* +signup(userDto: UserDto): Call<UserDto> - метод для регистрации в системе принимающий структуру пользователя, и возвращающий ее в случае успешной регистрации.
* +getinfo(id: uuid): Call<UserDto> - метод для получения информации о польозвателе, принимающий идентификатор пользователя (чаще хранящийся в JWT) и возвращающий всю информцию о пользователе.
* +order():<ResponseBody> - метод позволяющий забронировать автомобиль.

Описание методов CarApi (доступно администратору):

* +add(carDto: CarDto): Call<CarDto> - метод для добавления нового автомобиля, принимающий структуру автомобиля и возвращающий ее в случае успешного добавления.
* +get(id: uuid): Call<CarDto> - метод для получения информации об автомобиле, принимающий идентификатор автомобиля и возвращающий информацию об автомобиле.
* +update(id: uuid, carDto: CarDto): Call<CarDto> - метод для обновления информации об автомобиле, принимающий идентификатор автомобиля, и новую информацию в виде структуры.
* +remove(id: uuid): Call<CarDto> - метод для удаления автомобиля, принимающий идентификатор автомобиля и возвращающий информацию об удаленном автомобиле в случае успеха.

Теперь перейдём к диаграмме пакетов. Она показывает зависимости между пакетами модели клиентского приложения (Рисунок 4.2).

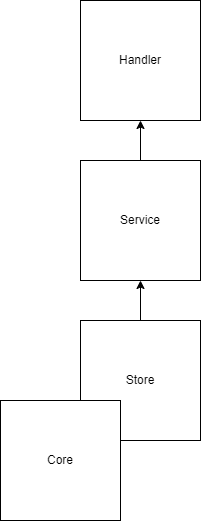


Рисунок 4.2 – Диаграмма пакетов

В пакетах расположены пакеты, которые дальнейшем будут использоваться для работы с данными, поступающими от сервера.