МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

Дисциплина: «Технологии разработки ПО»

**Отчет по лабораторной работе №14**

Создание диаграмм классов.

проект: «ПО для управление отелем»

Выполнили: ст. гр. 10701116 Мордик А.В.

Шантыр А.Ю.

Принял: доц. Сидорик В.В.

Минск 2020

**Теория**

**Классы.**

**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 четырех типов видимости:

* private (частный)
* protected (защищенный)
* public (общий)
* package or implementation (пакетный)

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.

**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.

**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).

Следует рассмотреть четыре различных типа операций.

**Операции реализации**(implementor operations) реализуют некоторую бизнес-функциональность. Такие операции можно найти, исследуя диаграммы взаимодействия. Диаграммы этого типа фокусируются на бизнес-функциональности, и каждое сообщение диаграммы скорее всего можно соотнести с опера­цией реализации.

**Операции управления**(manager operations) управляют созданием и разрушением объектов. В эту катего­рию попадают конструкторы и деструкторы классов. В среде Rose не требуется вручную создавать конструкторы и деструкторы классов. При генера­ции кода предоставляется возможность сделать это автоматически.

Атрибуты обычно бывают закрытыми или защищенными. Тем не менее другие классы иногда должны просматривать или изменять их значения. Для этого предназначены *операции доступа*(access operati­ons). Эти операции могут содержать любые правила и условия проверки, которые необходимо выполнить, прежде чем изменить атрибут.

**Вспомогательными**(helper operations) называются такие операции класса, которые необходимы ему для выполнения его ответственностей, но о которых другие классы не должны ничего знать. Это за­крытые и защищенные операции класса. Вспомогательные операции можно обнаружить на диаграммах после­довательности и кооперативных диаграммах. Часто такие операции являются рефлексивными сооб­щениями.

**Возвращаемым классом** (return class) операции называется тип данных ее результата. При определении возвращаемого класса можно использовать либо встроенные типы языка про­граммирования (такие, как string, char или integer), либо определенные в вашей модели классы.

Как и в случае других элементов модели, для классификации операций создаются их стереотипы. Существуют четыре наиболее распространенных стереотипа операций: *Implementor*(Реализация), *Manager*(Управляющая), *Access* (Доступ) и *Helper*(Вспомогательная). Назначение операциям стереотипов не требуется для генерации кода. Тем не менее они облегча­ют понимание модели.

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.

**Стереотипы классов**

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.

**Отношения между классами. 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  
**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». Рисунок 1.

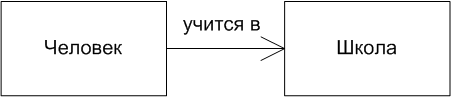


Рисунок 1 – Пример отношения ассоциации

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.

**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  
**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  
Третья связь – **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  
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

**Построение диаграммы классов, описание атрибутов, методов в классах и отношений между классами.**

Для создания диаграммы классов необходимо нажать правой клавишей мыши на логическое представление и в контекстном меню выбрать New – New class diagram.

Для добавления атрибутов класса необходимо кликнуть по нему правой клавишей мыши и в контекстном меню выбрать Open Specification. На вкладке Attributes на свободном месте нажать правой клавишей мыши и в контекстном меню выбрать Insert. Дважды нажав левой клавишей мыши на атрибут можно его редактировать.

Для добавления атрибутов класса необходимо кликнуть по нему правой клавишей мыши и в контекстном меню выбрать Open Specification. На вкладке Operations на свободном месте нажать правой клавишей мыши и в контекстном меню выбрать Insert. Дважды нажав левой клавишей мыши на операцию можно его редактировать.

Для добавления связи между классами необходимо выбрать тип связи на панели инструментов представленной на рисунке 2, нажать на один класс левой клавишей мыши и удерживая левую клавишу мыши протянуть связь до второго класса.



Рисунок 2 – Панель инструментов для диаграммы классов.

**Нотация диаграмм классов.**

Нотация диаграммы классов представлена на рисунках 3 и 4.

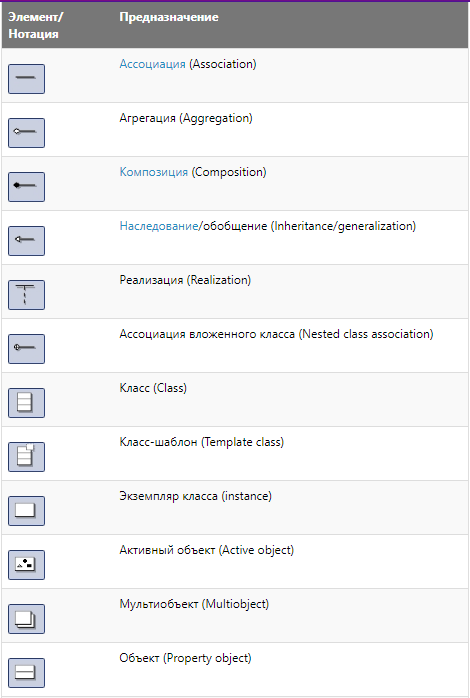


Рисунок 3 – Нотация диаграммы классов.

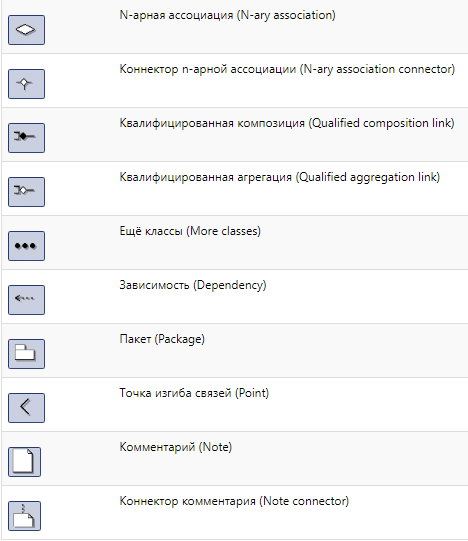


Рисунок 4 – Нотация диаграммы классов.

**Цель:**

Создание диаграмму классов для бизнес-процесса «Бронирование номера» в среде IBM Rational Rose.

**Задачи:**

1. Сгруппировать классы в соответствующие пакеты.
2. Добавить атрибуты классам и типы к операциям классов.
3. Определить связи между классами.

**Общее задание**

Диаграмма последовательности представлена на рисунке 5.

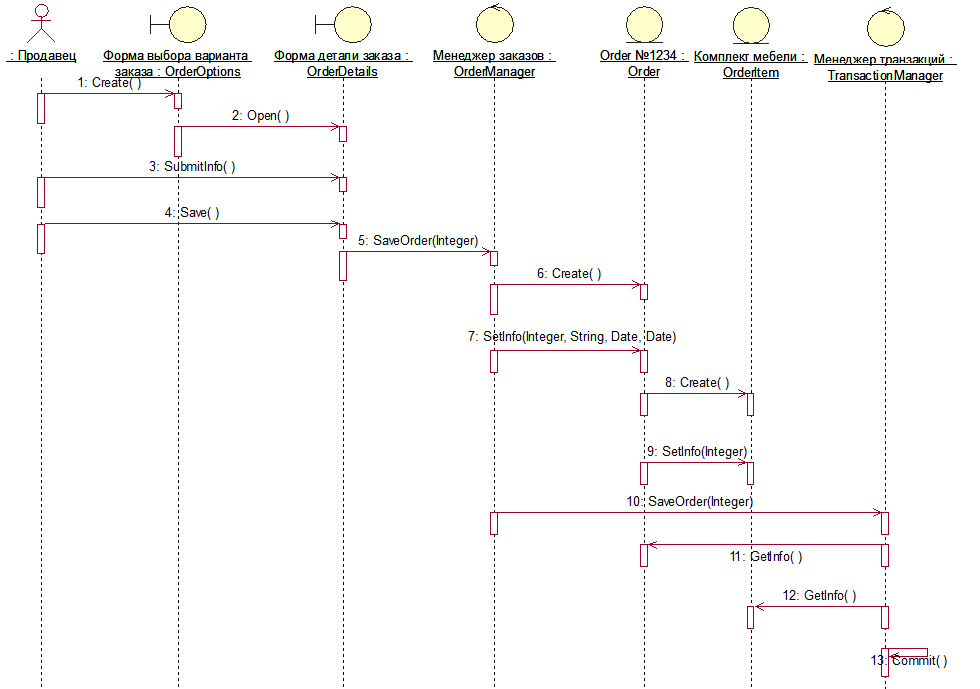


Рисунок 5 – Диаграмма последовательности для варианта использования «Ввести новый заказ»

**14.1 Создание пакетов и создание диаграммы классов**

Результат задания представлен на рисунке 6 и 7.

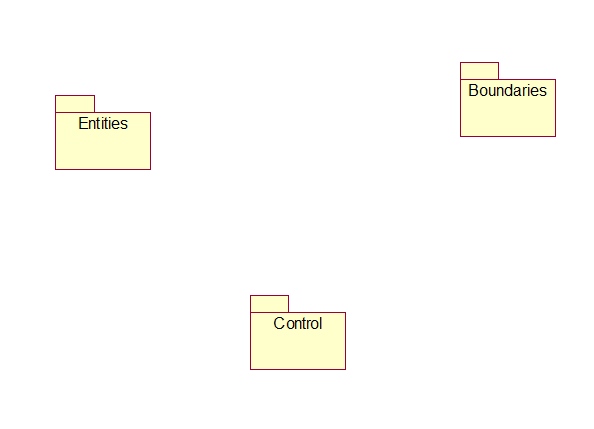


Рисунок 6 – Главная диаграмма классов системы обработки заказов

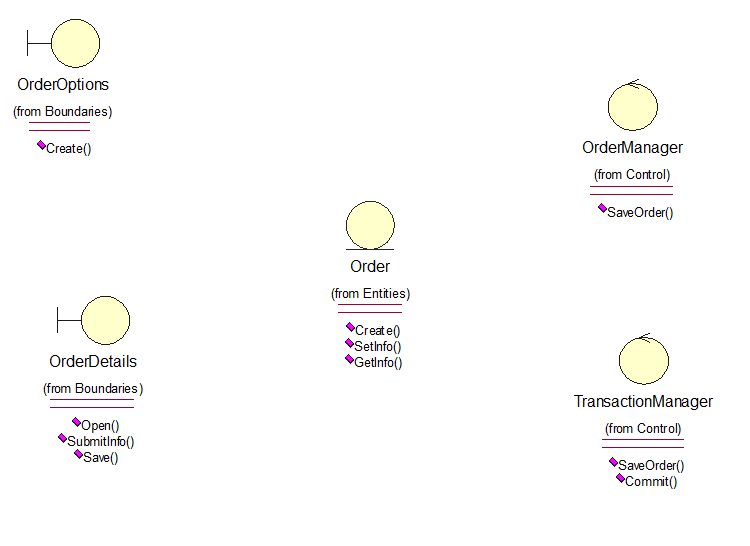


Рисунок 7 – Стереотипы классов для вариантов использования «Ввести новый заказ»

**14.2 Добавление атрибутов и операций**

Результат задания представлен на рисунке 8.

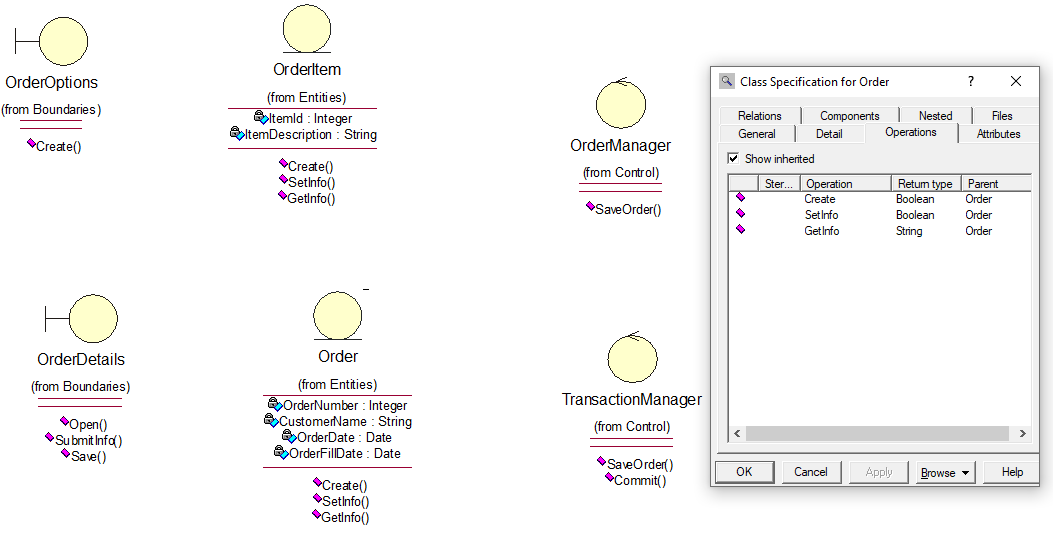


Рисунок 8 – Диаграмма классов варианта использования «Ввести новый заказ» с атрибутами и операциями

**14.3 Добавление связей между классами**

Результат задания представлен на рисунке 9.

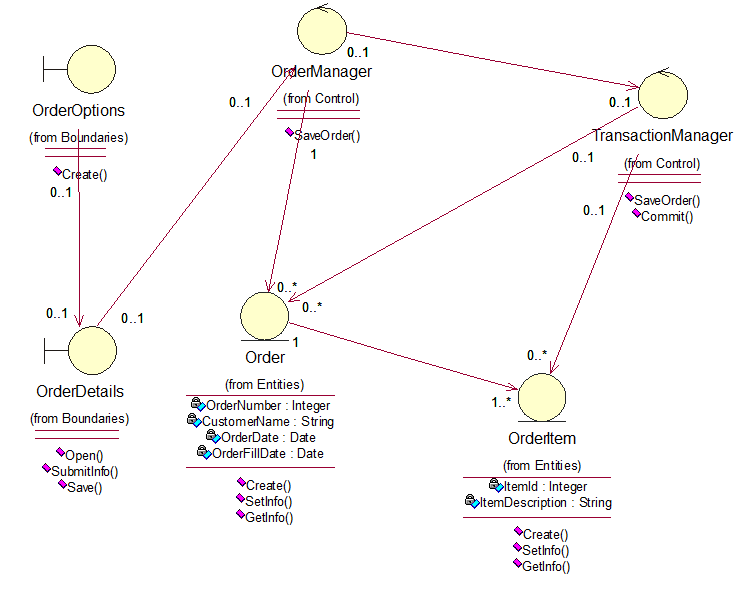


Рисунок 9 – Значения множественности для ассоциаций диаграммы классов варианта использования «Ввести новый заказ»

**Индивидуальное задание**

Диаграмма последовательности представлена на рисунке 10.

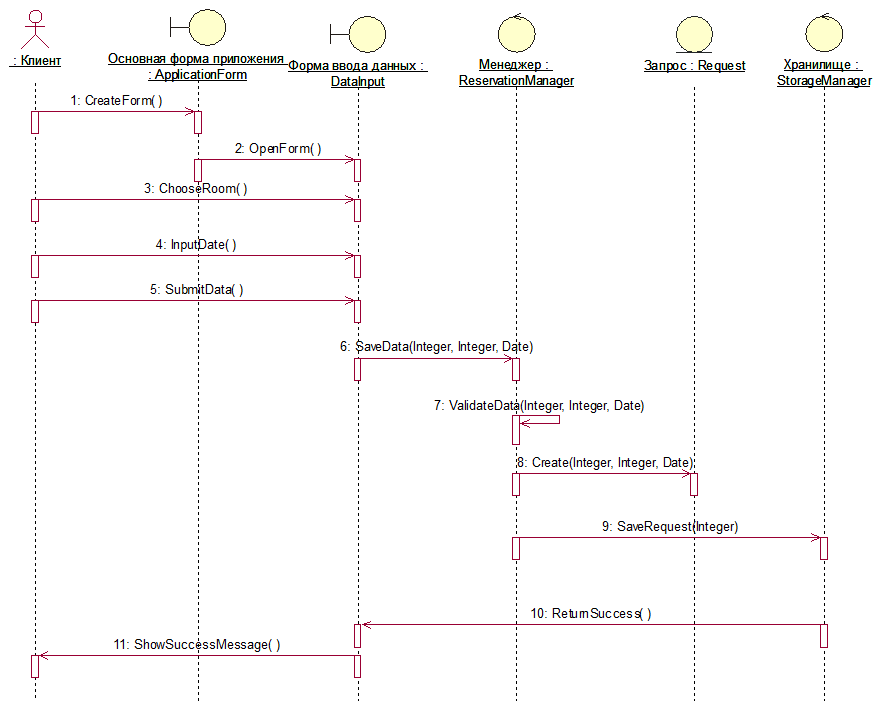


Рисунок 10 – Диаграмма последовательности бизнес-процесса «Бронирование номера»

**14.1 Создание пакетов и создание диаграммы классов**

Результат задания представлен на рисунке 11 и 12.

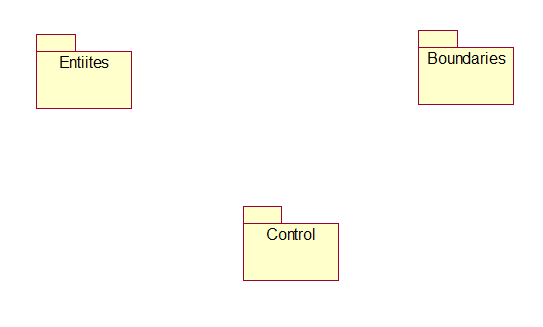
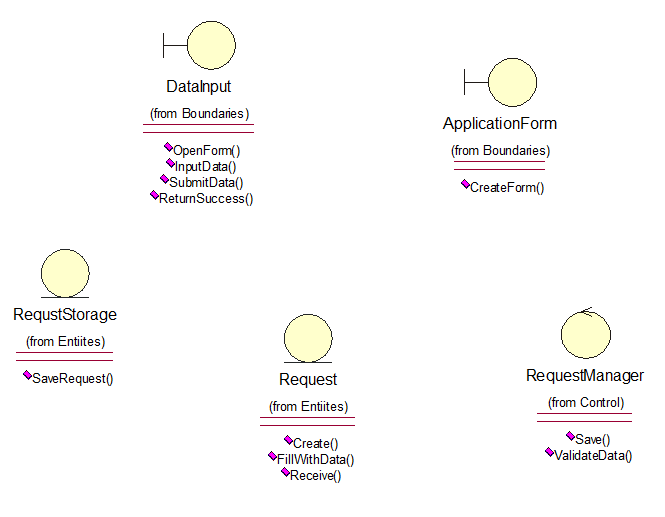


Рисунок 11 – Главная диаграмма классов системы пребывания в отел

Рисунок 12 – Стереотипы классов для вариантов использования «Бронирование номера»

**14.2 Добавление атрибутов и операций**

Результат задания представлен на рисунке 13.

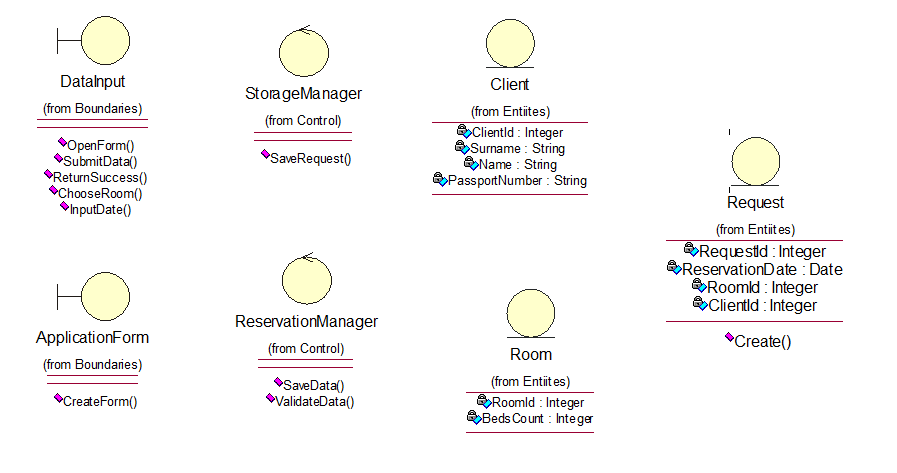


Рисунок 13 – Диаграмма классов варианта использования «Бронирование номера» с атрибутами и операциями

DataInput

Операции

* OpenForm() : Boolean
* SubmitData() : Boolean
* ReturnSuccess() : Boolean
* ChooseRoom() : RoomId
* InputDate() : Date

StorageManager

Операции

* SaveRequest(RequestId : Integer) : Boolean

Client

Атрибуты

* ClientId : Integer
* Surname : String
* Name : String
* PassportNumber : String

ApplicationForm

Операции

* CreateForm() : Boolean

ReservationManager

Операции

* SaveData(ClientId : Integer, RoomId : Integer, Date : Date) : Boolean
* ValidateData(ClientId : Integer, RoomId : Integer, Date : Date) : Boolean

Room

Атрибуты

* RoomId : Integer
* BedsCount : Integer

Request

Атрибуты

* RequestId : Integer
* ReservationDate : Date
* RoomId : Integer
* ClientId : Integer

Операции

* Create(ClientId : Integer, RoomId : Integer, ReservationDate : Date) : Integer

**14.3 Добавление связей между классами**

Результат задания представлен на рисунке 14.

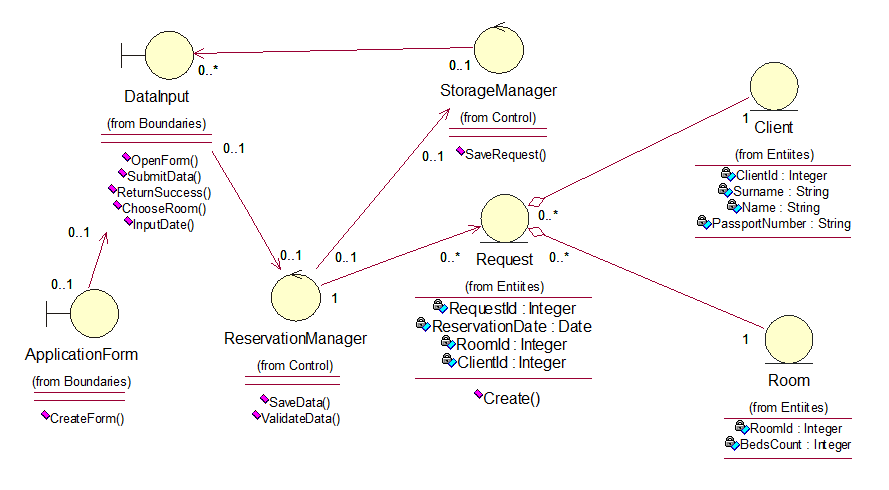


Рисунок 14 – Значения множественности для ассоциаций диаграммы классов варианта использования «Ввести новый заказ»

**Вывод**

В ходе выполнения лабораторной работы была разработана диаграммы классов для бизнес-процессов «Бронирование номера».

**Контрольные вопросы:**

1. Что такое класс? Требования к идентификатору класса. Пример идентификаторов.

**Класс** — в объектно-ориентированном программировании, представляет собой шаблон для создания [объектов](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), обеспечивающий начальные значения состояний: инициализация [полей-переменных](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B5_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B0) и реализация поведения функций или методов.

**Идентификатор** — это последовательность из одной или более латинских букв, цифр и символов подчеркивания, которая начинается с буквы или символа подчеркивания. Допускается любое число символов в идентификаторе. Примеры идентификаторов: User, Info123, \_Data, Main\_Window.

1. Пакеты Entities (Сущности), Boundaries (Границы) и Control (Управление) и их назначение.

**Пакет** в языке моделирования [UML](https://ru.wikipedia.org/wiki/UML) — основная группирующая сущность, позволяющая группировать элементы диаграммы для экономии места и обеспечения абстракции.

Назначение данных трех пакетов – разделение всех классов по трем разным категориям для удобства восприятия системы.

**Классы-сущности** (entity classes) отражают основные понятия (абстракции) предметной области и, как правило, содержат хранимую информацию. Данные классы располагаются в пакете Entities (Сущности).

**Граничные классы** (boundary classes) - это классы, которые расположены на границе системы и окружающей среды. Они включают все формы, отчеты, интерфейсы с аппаратурой (такой, как принтеры или сканеры) и интерфейсы с другими системами. Данные классы располагаются в пакете Boundaries.

**Управляющие классы** (control classes) отвечают за координацию действий других классов. Обычно у каждого варианта использования имеется один управляющий класс, контролирующий последовательность событий этого варианта использования. Данные классы располагаются в пакете Control.

1. Атрибуты и методы класса. Примеры на языке Java.

***Атрибут (attribute)*** — содержательная характеристика *класса*, описывающая множество значений, которые могут принимать отдельные объекты этого *класса*.

*Атрибут класса* служит для представления отдельного свойства или признака, который является общим для всех объектов данного *класса*. *Атрибуты класса* записываются во второй сверху секции прямоугольника *класса*. Эту секцию часто называют секцией *атрибутов*.

***Операция*** (метод) – это реализация метода класса. Класс может иметь любое число операций либо не иметь ни одной. Часто вызов операции объекта изменяет его атрибуты.

public class MyClass {

int x = 5;

public static void main(String[] args) {

MyClass **myObj1** = new MyClass(); // Object 1

MyClass **myObj2** = new MyClass(); // Object 2

System.out.println(myObj1.x);

System.out.println(myObj2.x);

}

}

1. Диаграмма классов и ее элементы.

**Диаграмма классов** (class diagram) служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений.

Диаграммы классов оперируют тремя видами сущностей UML:

· Структурные.

· Поведенческие.

· Аннотирующие.

**Структурные сущности** – 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 сущности** – 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 сущности** – это поясняющие части UML-моделей, иными словами, комментарии, которые можно применить для описания, выделения и пояснения любого элемента модели.

1. Модификаторы видимости и их назначение. Примеры на языке Java.

Модификаторы нужны, чтобы задать уровни доступа для классов, переменных, методов и конструкторов. Существует четыре доступа:

* Видимый в пакете (стоит по умолчанию и модификатор не требуются).
* Видимый только для класса (private).
* Видимый для всех (public).
* Видимый для пакета и всех подклассов (protected).

**Модификатор private** — методы, переменные и конструкторы, которые объявлены как private в Java могут быть доступны только в пределах самого объявленного класса.

Модификатор доступа private является наиболее ограничивающим уровнем доступа. Класс и интерфейсы не могут быть private.

Переменные, объявленные как private, могут быть доступны вне класса, если получающие их открытые (public) методы присутствуют в классе.

Использование модификатора private в Java является основным способом, чтобы скрыть данные.

Следующий класс использует контроль доступа private:

public class User {

private String info;

public String getInfo() {

return this.info;

}

public void setInfo(String info) {

this.info = info;

}

}

**Модификатор public** — класс, метод, конструктор, интерфейс и т.д. объявленные как public могут быть доступны из любого другого класса. Поэтому поля, методы, блоки, объявленные внутри public класса, могут быть доступны из любого класса, принадлежащего к "вселенной" Java.

Тем не менее, если к public классу в другом пакете мы пытаемся получить доступ, то public класс приходится импортировать.

Благодаря наследованию классов, в Java все публичные (public) методы и переменные класса наследуются его подклассами.

Следующая функция использует контроль доступа public:

public static void main(String[] arguments) {

}

**Модификатор protected** — переменные, методы и конструкторы, которые объявляются как protected в суперклассе, могут быть доступны только для подклассов в другом пакете или для любого класса в пакете класса protected.

Модификатор доступа protected в Java не может быть применен к классу и интерфейсам. Методы и поля могут быть объявлены как protected, однако методы и поля в интерфейсе не могут быть объявлены как protected.

Доступ protected дает подклассу возможность использовать вспомогательный метод или переменную, предотвращая неродственный класс от попыток использовать их.

Следующий родительский класс использует контроля доступа protected, чтобы его дочерний класс переопределил метод openSpeaker():

class AudioPlayer {

protected boolean openSpeaker(Speaker sp) {

// детали реализации

}

}

class StreamingAudioPlayer {

boolean openSpeaker(Speaker sp) {

// детали реализации

}

}

При этом, если мы определим метод openSpeaker() как protected, то он не будет доступен из любого другого класса, кроме AudioPlayer. Если мы определим его как public, то он станет доступным всем. Но наше намерение состоит в том, чтобы раскрыть этот метод только подклассу, вот почему мы использовали модификатор protected.

**Список литературы:**

1. Методические указания к лабораторной работе №14 – [Электронный ресурс] – Google Drive –<https://drive.google.com/drive/u/0/folders/1Gl489J1HVRTDCcomrL0nTXBObMHEySBHh>– Дата обращения 02.03.2020
2. Диаграмма классов – [Электронный ресурс] – Flexberry –<https://flexberry.github.io/ru/fd_class-diagram.html> – Дата обращения 02.03.2020
3. UML. Диаграмма классов – [Электронный ресурс] – intuit –<https://www.intuit.ru/studies/courses/32/32/lecture/1008?page=2> – Дата обращения 02.03.2020
4. Java classes – [Электронный ресурс] – w3schools –<https://www.w3schools.com/java/java_classes.asp> – Дата обращения 02.03.2020

**Глоссарий:**

**Диаграмма классов (class diagram)** служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о временных аспектах функционирования системы. С этой точки зрения диаграмма классов является дальнейшим развитием концептуальной модели проектируемой системы.

**Класс (class)** в языке UML служит для обозначения множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов. Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции.

**Атрибут (свойство, data member, class field, instance variable)** – это именованное свойство класса, описывающее диапазон значений, которые может принимать экземпляр атрибута.

**Операция (operation**) представляет собой некоторый сервис, предоставляющий каждый экземпляр класса по определенному требованию. Совокупность операций характеризует функциональный аспект поведения класса.

**Отношение агрегации (aggregation)** имеет место между несколькими классами в том случае, если один из классов представляет собой некоторую сущность, включающую в себя в качестве составных частей другие сущности.

**Отношение композиции (composition)** является частным случаем отношения агрегации. Это отношение служит для выделения специальной формы отношения "часть-целое", при которой составляющие части в некотором смысле находятся внутри целого. Специфика взаимосвязи между ними заключается в том, что части не могут выступать в отрыве от целого, т. е. с уничтожением целого уничтожаются и все его составные части.

**Наследование (inheritance)** – это отношение типа «общее-частное». Позволяет определить такое отношение между классами, когда один класс обладает поведением и структурой ряда других классов.

**Стереотип (stereotype)** класса – это элемент расширения словаря UML, который обозначает отличительные особенности в использовании класса. Стереотип имеет название, которое задается в виде текстовой строки. При изображении класса на диаграмме стереотип показывается в верхней части класса в двойных угловых скобках.