**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ**

**Факультет непрерывного и дистанционного обучения**

# Специальность: Автоматизированные системы обработки информации

**ЛАБОРАТОРНЫЙ ПРАКТИКУМ**

**ПО ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ  
ПРОГРАММИРОВАНИЕ И ПРОЕКТИРОВАНИЕ**

***Соболевского Дмитрия Александровича***

***Группа: 590651***

***Зачетная книжка: ‎000623-28***

***Электронный адрес:*** [*sobolevskidmitry@gmail.com*](mailto:sobolevskidmitry@gmail.com) ***/ BSUIR\sda***

**Лабораторная работа №1. Создание диаграммы вариантов использования**

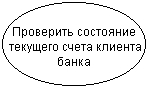
***Цель работы:***

* изучение диаграмм вариантов использования,
* изучение их применения в процессе постановки задачи.

***Теория:***

Конструкция или стандартный элемент языка UML вариант использования применяется для спецификации общих особенностей поведения системы или любой другой сущности предметной области без рассмотрения внутренней структуры этой сущности. Каждый вариант использования определяет последовательность действий, которые должны быть выполнены проектируемой системой при взаимодействии ее с соответствующим актером. Диаграмма вариантов может дополняться пояснительным текстом, который раскрывает смысл или семантику составляющих ее компонентов. Такой пояснительный текст получил название примечания или сценария.

Отдельный вариант использования обозначается на диаграмме эллипсом, внутри которого содержится его краткое название или имя в форме глагола с пояснительными словами (рис. 4.1).



**Рис. 4.1.** Графическое обозначение варианта использования

Цель варианта использования заключается в том, чтобы определить законченный аспект или фрагмент поведения некоторой сущности без раскрытия внутренней структуры этой сущности. В качестве такой сущности может выступать исходная система или любой другой элемент модели, который обладает собственным поведением, подобно подсистеме или классу в модели системы.

Каждый вариант использования соответствует отдельному сервису, который предоставляет моделируемую сущность или систему по запросу пользователя (актера), т. е. определяет способ применения этой сущности. Сервис, который инициализируется по запросу пользователя, представляет собой законченную последовательность действий. Это означает, что после того как система закончит обработку запроса пользователя, она должна возвратиться в исходное состояние, в котором готова к выполнению следующих запросов.

Варианты использования описывают не только взаимодействия между пользователями и сущностью, но также реакции сущности на получение отдельных сообщений от пользователей и восприятие этих сообщений за пределами сущности. Варианты использования могут включать в себя описание особенностей способов реализации сервиса и различных исключительных ситуаций, таких как корректная обработка ошибок системы. Множество вариантов использования в целом должно определять все возможные стороны ожидаемого поведения системы. Для удобства множество вариантов использования может рассматриваться как отдельный пакет.

С системно-аналитической точки зрения варианты использования могут применяться как для спецификации внешних требований к проектируемой системе, так и для спецификации функционального поведения уже существующей системы. Кроме этого, варианты использования неявно устанавливают требования, определяющие, как пользователи должны взаимодействовать с системой, чтобы иметь возможность корректно работать с предоставляемыми данной системой сервисами!

**Примечание**

Каждый выполняемый вариантом использования метод реализуется как неделимая транзакция, т. е. выполнение сервиса не может быть прервано никаким другим экземпляром варианта использования.

Применение вариантов использования на всех уровнях диаграммы позволяет не только достичь требуемого уровня унификации обозначений для представления функциональности подсистем и системы в целом, но и является мощным средством последовательного уточнения требований к проектируемой системе на основе полууровневого спуска от пакетов системы к операциям классов. С другой стороны, модификация отдельных операций класса может оказать обратное влияние на уточнение сервиса соответствующего варианта использования, т. е. реализовать эффект обратной связи с целью уточнения спецификаций или требований на уровне пакетов системы.

В метамодели UML вариант использования является подклассом классификатора, который описывает последовательности действий, выполняемых отдельным экземпляром варианта использования. Эти действия включают изменения состояния и взаимодействия со средой варианта использования. Эти последовательности могут описываться различными способами, включая такие, как графы деятельности и автоматы.

Примерами вариантов использования могут являться следующие действия: проверка состояния текущего счета клиента, оформление заказа на покупку товара, получение дополнительной информации о кредитоспособности клиента, отображение графической формы на экране монитора и другие действия.

  4.2. Актеры

Актер представляет собой любую внешнюю по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач. При этом актеры служат для обозначения согласованного множества ролей, которые могут играть пользователи в процессе взаимодействия с проектируемой системой. Каждый актер может рассматриваться как некая отдельная роль относительно конкретного варианта использования. Стандартным графическим обозначением актера на диаграммах является фигурка "человечка", под которой записывается конкретное имя актера (рис. 4.2).

http://khpi-iip.mipk.kharkiv.edu/library/case/leon/gl4/gl4-2.jpg

**Рис. 4.2.** Графическое обозначение актера

В некоторых случаях актер может обозначаться в виде прямоугольника класса с ключевым словом "актер" и обычными составляющими элементами класса. Имена актеров должны записываться заглавными буквами и следовать рекомендациям использования имен для типов и классов модели. При этом символ отдельного актера связывает соответствующее описание актера с конкретным именем. Имена абстрактных актеров, как и других абстрактных элементов языка UML, рекомендуется обозначать курсивом.

**Примечание**

Имя актера должно быть достаточно информативным с точки зрения семантики. Вполне подходят для этой цели наименования должностей в компании (например, продавец, кассир, менеджер, президент). Не рекомендуется давать актерам имена собственные (например, "О.Бендер") или моделей конкретных устройств (например, "маршрутизатор Cisco 3640"), даже если это с очевидностью следует из контекста проекта. Дело в том, что одно и то же лицо может выступать в нескольких ролях и, соответственно, обращаться к различным сервисам системы. Например, посетитель банка может являться как потенциальным клиентом, и тогда он востребует один из его сервисов, а может быть и налоговым инспектором или следователем прокуратуры. Сервис для последнего может быть совершенно исключительным по своему характеру.

Примерами актеров могут быть: клиент банка, банковский служащий, продавец магазина, менеджер отдела продаж, пассажир авиарейса, водитель автомобиля, администратор гостиницы, сотовый телефон и другие сущности, имеющие отношение к концептуальной модели соответствующей предметной области.

**Примечание**

В метамодели актер является подклассом классификатора. Актеры могут взаимодействовать с множеством вариантов использования и иметь множество интерфейсов, каждый из которых может представлять особенности взаимодействия других элементов с отдельными актерами.

Актеры используются для моделирования внешних по отношению к проектируемой системе сущностей, которые взаимодействуют с системой и используют ее в качестве отдельных пользователей. В качестве актеров могут выступать другие системы, подсистемы проектируемой системы или отдельные классы. Важно понимать, что каждый актер определяет некоторое согласованное множество ролей, в которых могут выступать пользователи данной системы в процессе взаимодействия с ней. В каждый момент времени с системой взаимодействует вполне определенный пользователь, при этом он играет или выступает в одной из таких ролей. Наиболее наглядный пример актера - конкретный пользователь системы со своими собственными параметрами аутентификации.

Любая сущность, которая согласуется с подобным неформальным определением актера, представляет собой экземпляр или пример актера. Для моделируемой системы актерами могут быть как субъекты-пользователи, так и другие системы. Поскольку пользователи системы всегда являются внешними по отношению к этой системе, то они всегда представляются в виде актеров.

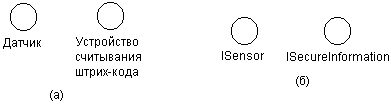
Так как в общем случае актер всегда находится вне системы, его внутренняя структура никак не определяется. Для актера имеет значение только его внешнее представление, т. е. то, как он воспринимается со стороны системы. Актеры взаимодействуют с системой посредством передачи и приема сообщений от вариантов использования. Сообщение представляет собой запрос актером сервиса от системы и получение этого сервиса. Это взаимодействие может быть выражено посредством ассоциаций между отдельными актерами и вариантами использования или классами. Кроме этого, с актерами могут быть связаны интерфейсы, которые определяют, каким образом другие элементы модели взаимодействуют с этими актерами.

Два и более актера могут иметь общие свойства, т. е. взаимодействовать с одним и тем же множеством вариантов использования одинаковым образом. Такая общность свойств и поведения представляется в виде рассматриваемого ниже отношения обобщения с другим, возможно, абстрактным актером, который моделирует соответствующую общность ролей. Совокупность отношений, которые могут присутствовать на диаграмме вариантов использования, будет рассмотрена ниже в данной главе.

4.3. Интерфейсы

Интерфейс (interface) служит для спецификации параметров модели, которые видимы извне без указания их внутренней структуры. В языке UML интерфейс является классификатором и характеризует только ограниченную часть поведения моделируемой сущности. Применительно к диаграммам вариантов использования, интерфейсы определяют совокупность операций, которые обеспечивают необходимый набор сервисов или функциональности для актеров. Интерфейсы не могут содержать ни атрибутов, ни состояний, ни направленных ассоциаций. Они содержат только операции без указания особенностей их реализации. Формально интерфейс эквивалентен абстрактному классу без атрибутов и методов с наличием только абстрактных операций.

На диаграмме вариантов использования интерфейс изображается в виде маленького круга, рядом с которым записывается его имя (рис. 4.3, а). В качестве имени может быть существительное, которое характеризует соответствующую информацию или сервис (например, "датчик", "сирена", "видеокамера"), но чаще строка текста (например, "запрос к базе данных", "форма ввода", "устройство подачи звукового сигнала"). Если имя записывается на английском, то оно должно начинаться с заглавной буквы I, например, ISecurelnformation, ISensor (рис. 4.3, б).

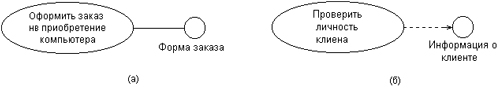


**Рис. 4.3.** Графическое изображение интерфейсов на диаграммах вариантов использования

**Примечание**

Имена интерфейсов подчиняются общим правилам наименования компонентов языка UML, т. е. имя может состоять из любого числа букв, цифр и некоторых знаков препинания, таких как двойное двоеточие "::". Последний символ используется для более сложных имен, включающих в себя не только имя самого интерфейса (после знака), но и имя сущности, которая включает в себя данный интерфейс (перед знаком). Примерами таких имен являются: "Сеть предприятия сервер" для указания на сервер сети предприятия или "Система аутентификации клиентов::форма ввода пароля".

Графический символ отдельного интерфейса может соединяться на диаграмме сплошной линией с тем вариантом использования, который его поддерживает. Сплошная линия в этом случае указывает на тот факт, что связанный с интерфейсом вариант использования должен реализовывать все операции, необходимые для данного интерфейса, а возможно и больше (рис. 4.4, а). Кроме этого, интерфейсы могут соединяться с вариантами использования пунктирной линией со стрелкой (рис. 4.4, б), означающей, что вариант использования предназначен для спецификации только того сервиса, который необходим для реализации данного интерфейса.



**Рис. 4.4.** Графическое изображение взаимосвязей интерфейсов с вариантами использования

С системно-аналитической точки зрения интерфейс не только отделяет спецификацию операций системы от их реализации, но и определяет общие границы проектируемой системы. В последующем интерфейс может быть уточнен явным указанием тех операций, которые специфицируют отдельный аспект поведения системы. В этом случае он изображается в форме прямоугольника класса с ключевым словом "interface" в секции имени, с пустой секцией атрибутов и с непустой секцией операций. Однако подобное графическое представление используется на диаграммах классов или диаграммах, характеризующих поведение моделируемой системы.

Важность интерфейсов заключается в том, что они определяют стыковочные узлы в проектируемой системе, что совершенно необходимо для организации коллективной работы над проектом. Более того, спецификация интерфейсов способствует "безболезненной" модификации уже существующей системы при переходе на новые технологические решения. В этом случае изменению подвергается только реализация операций, но никак не функциональность самой системы. А это обеспечивает совместимость последующих версий программ с первоначальными при спиральной технологии разработки программных систем.

4.4. Примечания

Примечания (notes) в языке UML предназначены для включения в модель произвольной текстовой информации, имеющей непосредственное отношение к контексту разрабатываемого проекта. В качестве такой информации могут быть комментарии разработчика (например, дата и версия разработки диаграммы или ее отдельных компонентов), ограничения (например, на значения отдельных связей или экземпляры сущностей) и помеченные значения. Применительно к диаграммам вариантов использования примечание может носить самую общую информацию, относящуюся к общему контексту системы.

Графически примечания обозначаются прямоугольником с "загнутым" верхним правым уголком (рис. 4.5). Внутри прямоугольника содержится текст примечания. Примечание может относиться к любому элементу диаграммы, в этом случае их соединяет пунктирная линия. Если примечание относится к нескольким элементам, то от него проводятся, соответственно, несколько линий. Разумеется, примечания могут присутствовать не только на диаграмме вариантов использования, но и на других канонических диаграммах.



**Рис. 4.5.** Примеры примечаний в языке UML

Если в примечании указывается ключевое слово "constraint", то данное примечание является ограничением, налагаемым на соответствующий элемент модели, но не на саму диаграмму. При этом запись ограничения заключается в фигурные скобки и должна соответствовать правилам правильного построения выражений языка ОСL. Более подробно язык объектных ограничений и примеры его использования будут рассмотрены в приложении. Однако для диаграмм вариантов использования ограничения включать в модели не рекомендуется, поскольку они достаточно жестко регламентируют отдельные аспекты системы. Подобная регламентация противоречит неформальному характеру общей модели системы, в качестве которой выступает диаграмма вариантов использования.

***Упражнения:***

**Создать диаграммы Вариантов Использования, вариантов использования и Действующих лиц**

1. Дважды щелкните на Главной диаграмме Вариантов Использования (Main) в броузере, чтобы открыть ее.
2. С помощью кнопки Use Case (Вариант Использования) панели инструментов поместите на диаграмму новый вариант использования.
3. Назовите этот новый вариант использования "Ввести новый заказ".
4. Повторите этапы 2 и 3, чтобы поместить на диаграмму остальные варианты использования: Изменить существующий заказ, Напечатать инвентарную опись, Обновить инвентарную опись, Оформить заказ, Отклонить заказ
5. С помощью кнопки Actor (Действующее лицо) панели инструментов поместите на диаграмму новое действующее лицо.
6. Назовите его "Продавец"
7. Повторите шаги 5 и 6, поместив на диаграмму остальных действующих лиц: Управляющий магазином, Клерк магазина, Бухгалтерская система

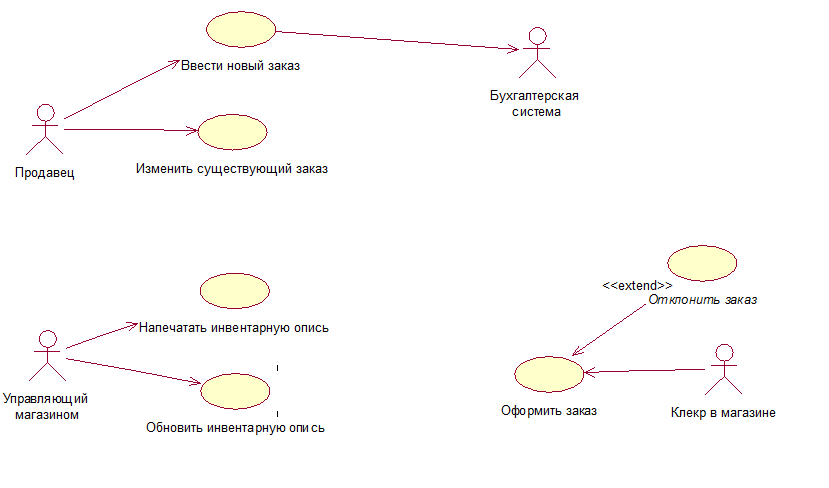


Рис. 1. Диаграмма Вариантов Использования для системы обработки заказов.

**Указать абстрактные варианты использования**

1. Щелкните правой кнопкой мыши на варианте использования "Отклонить заказ" на диаграмме.
2. В открывшемся меню выберите пункт Open Specification (Открыть спецификацию).
3. Пометьте контрольный переключатель Abstract (Абстрактный), чтобы сделать этот вариант использования абстрактным.

**Добавить ассоциации**

1. С помощью кнопки Unidirectional Association (Однонаправленная ассоциация) панели инструментов нарисуйте ассоциацию между действующим лицом Продавец и вариантом использования "Ввести новый заказ".
2. Повторите этот этап, чтобы поместить на диаграмму остальные ассоциации.

**Добавить связь расширения**

1. С помощью кнопки Generalization панели инструментов нарисуйте связь между вариантом использования "Отклонить заказ" и вариантом использования "Оформить заказ". Стрелка должна протянуться от первого варианта использования ко второму. Связь расширения означает, что вариант использования "Отклонить заказ" при необходимости дополняет функциональные возможности варианта использования "Оформить заказ".
2. Щелкните правой кнопкой мыши на новой связи между вариантами использования "Отклонить заказ" и "Оформить заказ".
3. В открывшемся меню выберите пункт Open Specification (Открыть спецификацию).
4. В раскрывающемся списке стереотипов введите слово extends (расширение), затем нажмите ОК.
5. Слово <<extends>> появится на линии данной связи.

**Добавить описания к вариантам использования**

1. Выделите в броузере вариант использования "Ввести новый заказ".
2. В окне документации введите следующее описание к этому варианту использования: Этот вариант использования дает клиенту возможность ввести новый заказ в систему.
3. С помощью окна документации введите описания ко всем остальным вариантам использования.

**Добавить описания к действующему лицу**

1. Выделите в броузере действующее лицо Продавец
2. В окне документации введите для этого действующего лица следующее описание: Продавец - это служащий, доставляющий и старающийся продать продукцию.
3. С помощью окна документации введите описания к оставшимся действующим лицам.

**Прикрепление файла к варианту использования**

1. Для описания главного потока событий варианта использования "Ввести новый заказ" создайте файл OrderFlow.doc, содержащий следующий текст:
2. *Продавец выбирает пункт "Создать новый заказ" из имеющегося меню.*
3. *Система выводит форму "Подробности заказа".*
4. *Продавец вводит номер заказа, заказчика и то, что заказано.*
5. *Продавец сохраняет заказ.*
6. *Система создает новый заказ и сохраняет его в базе данных.*
7. Щелкните правой кнопкой мыши на варианте использования "Ввести новый заказ".
8. В открывшемся меню выберите пункт Open Specification (Открыть спецификацию)
9. Перейдите на вкладку файлов.
10. Щелкните правой кнопкой мыши на белом поле и из открывшегося меню выберите пункт Insert File (Ввести файл).
11. Укажите файл OpenFlow.doc и нажмите на кнопку Open (Открыть), чтобы прикрепить файл к варианту использования.

**Лабораторная работа №2. Создание диаграмм взаимодействия**

***Цель работы:***

* изучение диаграмм взаимодействия
* изучение их применения в процессе проектирования.

***Теория:***

Диаграммы взаимодействия являются моделями, описывающими поведение взаимодействующих групп объектов.

Как правило, диаграмма взаимодействия охватывает поведение только одного варианта использования. На такой диаграмме отображается ряд объектов и те сообщения, которыми они обмениваются между собой в рамках данного варианта использования.

Существует два вида диаграмм взаимодействия: диаграммы последовательности (sequence diagrams) и кооперативные диаграммы (collaboration diagrams).

## *Создание диаграмм Взаимодействия*

Создайте диаграмму Последовательности и Кооперативную диаграмму, отражающую ввод нового заказа в систему обработки заказов. Готовая диаграмма Последовательности должна выглядеть как на рисунке 5.

Это только одна из диаграмм, необходимых для моделирования варианта использования "Ввести новый заказ". Она соответствует успешному варианту хода событий. Для описания того, что случится, если возникнет ошибка, или если пользователь выберет другие действия из предложенных, придется разработать другие диаграммы. Каждый альтернативный поток варианта использования может быть промоделирован с помощью своих собственных диаграмм Взаимодействия.

***Упражнения:***

### Настройка

1. В меню модели выберите пункт Tools > Options (Инструменты > Параметры).
2. Перейдите на вкладку диаграмм.
3. Контрольные переключатели Sequence Numbering, Collaboration Numbering и Focus of Control должны быть помечены.
4. Нажмите ОК, чтобы выйти из окна параметров.

### Создание диаграммы Последовательности

1. Щелкните правой кнопкой мыши на Логическом представлении броузера.
2. В открывшемся меню выберите пункт New > Sequence Diagram.
3. Назовите новую диаграмму "Ввод заказа".
4. Дважды щелкните на ней, чтобы открыть ее.

### Добавление на диаграмму действующего лица и объектов

1. Перетащите действующее лицо Продавец (Salesperson) с броузера на диаграмму.
2. На панели инструментов нажмите кнопку Object (Объект).
3. Щелкните мышью в верхней части диаграммы, чтобы поместить туда новый объект.
4. Назовите объект "Order Options Form -- Выбор варианта заказа".
5. Повторите этапы 3 и 4, чтобы поместить на диаграмму все остальные объекты:

# "Order Detail Form" -- Форма Детали заказа

# "Order N1234" -- Заказ №1234.

### Добавление сообщений на диаграмму

1. На панели инструментов нажмите кнопку Object Message (Сообщение объекта).
2. Проведите мышью от линии жизни актера Продавец к линии жизни объекта Выбор варианта заказа.
3. Выделив сообщение, введите его имя "Create New Order" -- Создать новый заказ.
4. Повторите этапы 2 и 3, чтобы поместить на диаграмму дополнительные сообщения:

# Open form -- Открыть форму (между Выбором варианта заказа и Деталями заказа)

# Enter order number, customer, order items -- Ввести номер заказа, заказчика и число заказываемых предметов (между Продавцом и Деталями заказа)

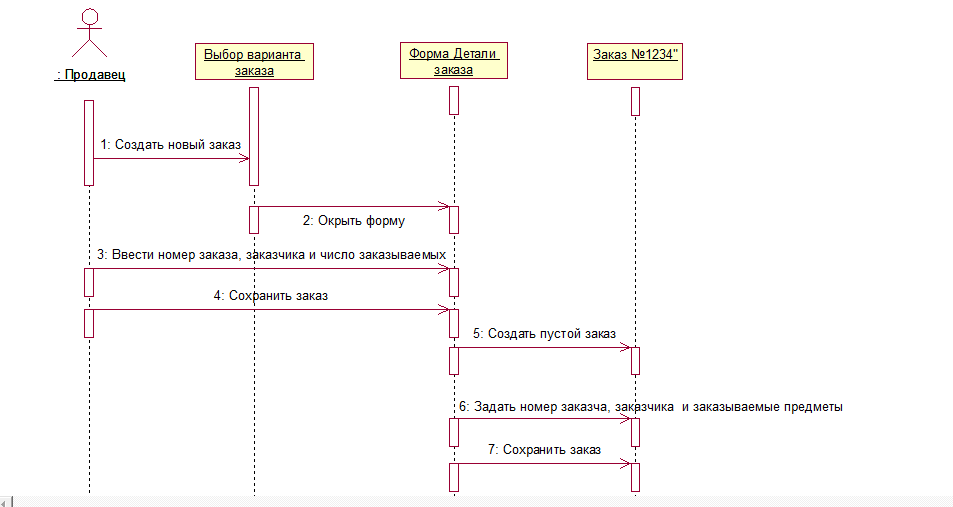
# Save the order -- Сохранить заказ (между Продавцом и Деталями заказа)

# Create new, blank order -- Создать пустой заказ (между Деталями заказа и Заказом №1234)

# Set the order number, customer, order items -- Ввести номер заказа, заказчика и число заказываемых предметов (между Деталями заказа и Заказом №1234).

# Save the order -- Сохранить заказ (между Деталями заказа и Заказом №1234)

Мы завершили первый этап работы. Готовая диаграмма Последовательности представлена на рисунке 2.



Теперь надо позаботиться об управляющих объектах и взаимодействии с базой данных. Как видно из диаграммы, объект Детали заказа имеет множество ответственностей, с которыми лучше всего мог бы справиться управляющий объект. Кроме того, новый заказ должен сохранять себя в базе данных сам. Вероятно, эту обязанность лучше было бы переложить на другой объект.

### Добавление на диаграмму дополнительных объектов

1. На панели инструментов нажмите кнопку Object.
2. Щелкните мышью между объектами Детали заказа и Заказ №1234, чтобы поместить туда новый объект.
3. Введите имя объекта - Order Manager (Управляющий заказами).
4. На панели инструментов нажмите кнопку Object.
5. Новый объект расположите справа от Заказа №1234.
6. Введите его имя - Transaction Manager (Управляющий транзакциями).

### Назначение ответственностей объектам

1. Выделите сообщение 5 (Создать пустой заказ).
2. Нажмите комбинацию клавиш CTRL + D, чтобы удалить это сообщение.
3. Повторите этапы 1 и 2, чтобы удалить два последних сообщения:

# Вести номер заказа, заказчика и число заказываемых предметов

# Сохранить заказ

1. На панели инструментов нажмите кнопку Object Message.
2. Поместите на диаграмму новое сообщение, расположив его под сообщением 4 между Деталями заказа и Управляющим заказами.
3. Назовите его Save the order (Сохранить заказ).
4. Повторите этапы 4 - 6, добавив сообщения с шестого по девятое и назвав их:

# Create new, blank order (Создать новый заказ) - между Управляющим заказами и Заказом №1234.

# Set the order number, customer, order items (Вести номер заказа, заказчика и число заказываемых предметов) - между Управляющим заказами и Заказом №1234.

# Save the order (Сохранить заказ) - между Управляющим заказами и Управляющим транзакциями.

# Collect order information (Информация о заказе) - между Управляющим транзакциями и Заказом №1234.

1. На панели инструментов нажмите кнопку Message to Self (Сообщение себе).
2. Щелкните на линии жизни объекта Управляющий транзакциями ниже сообщения 9, добавив туда рефлексивное сообщение. Назовите его Save the order information to the database (Сохранить информацию о заказе в базе данных).

### Соотнесение объектов с классами

1. Щелкните правой кнопкой мыши на объекте Выбор варианта заказа.
2. В открывшемся меню выберите пункт Open Specification (Открыть спецификацию).
3. В раскрывающемся списке классов выберите пункт <New> (Создать). Появится окно спецификации классов.
4. В поле имени введите имя OrderOptions (Выбор заказа).
5. Щелкните на кнопке ОК. Вы вернетесь к окну спецификации объекта.
6. В списке классов выберите теперь класс OrderOptions.
7. Щелкните на кнопке ОК, чтобы вернуться к диаграмме. Теперь объект называется Order Options Form : OrderOptions (Выбор варианта заказа : OrderOptions).
8. Для соотнесения остальных объектов с классами повторите этапы с 1 по 7:

# Класс OrderDetail соотнесите с объектом Детали заказа.

# Класс OrderMgr - с объектом Управляющий заказами.

# Класс Order - с объектом Заказ №1234.

# Класс TransactionMgr - с объектом Управляющий транзакциями.

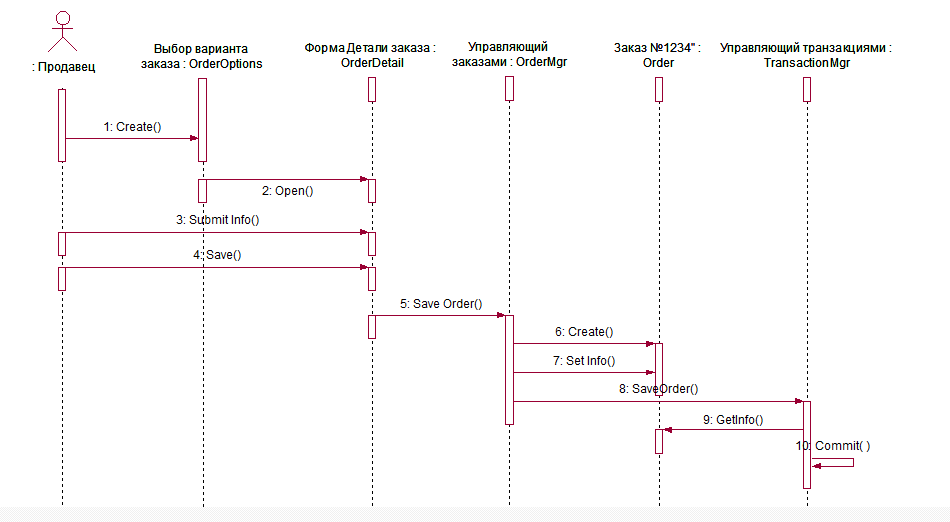


Рис 3. Диаграмма Последовательности с показанными на ней операциями.

### Соотнесение сообщений с операциями

1. Щелкните правой кнопкой на сообщении 1, Создать новый заказ.
2. В открывшемся меню выберите пункт <new operation> (создать операцию). Появится окно спецификации операции.
3. В поле имени введите имя операции - Create (Создать).
4. Нажмите на кнопку ОК, чтобы закрыть окно спецификации операции и вернуться на диаграмму.
5. Еще раз щелкните правой кнопкой мыши на сообщении 1.
6. В открывшемся меню выберите новую операцию Create().
7. Повторите сообщения с 1 по 6, пока не соотнесете с операциями все остальные сообщения:

# Сообщение 2:Открыть соотнесите с операцией Open()

# Сообщение 3: Ввести номер заказа, заказчика и число заказываемых предметов - с операцией SubmitInfo().

# Сообщение 4:Сохранить заказ - с операцией Save().

# Сообщение 5:Сохранить заказ - с операцией SaveOrder().

# Сообщение 6:Создать пустой заказ - с операцией Create().

# Сообщение 7: Ввести номер заказа, заказчика и число заказываемых предметов - с операцией SetInfo().

# Сообщение 8:Сохранить заказ - с операцией SaveOrder().

# Сообщение 9:Информация о заказе - с операцией GetInfo().

# Сообщение 10:Сохранить информацию о заказе в базе данных - с операцией Commit.

### Создание Кооперативной диаграммы

Для создания Кооперативной диаграммы достаточно просто нажать клавишу F5 или, если вы хоти сами проделать все требуемые операции, воспользуйтесь приводимым далее планом.

1. Щелкните правой кнопкой мыши на Логическом представлении в броузере.
2. В открывшемся меню выберите пункт New > Collaboration Diagram.
3. Назовите эту диаграмму Ввод заказа.
4. Щелкните на ней дважды, чтобы открыть ее.

### Добавление действующего лица и объектов на диаграмму

1. Перетащите действующее лицо Продавец (Salesperson) с броузера на диаграмму.
2. На панели инструментов нажмите кнопку Object (Объект).
3. Щелкните мышью где-нибудь внутри диаграммы, чтобы поместить туда новый объект.
4. Назовите объект "Order Options Form" -- Выбор варианта заказа.
5. Повторите этапы 3 и 4, чтобы поместить на диаграмму все остальные объекты:

# "Order Detail Form" -- Форма Детали заказа

# "Order N1234" -- Заказ №1234.

### Добавление сообщений на диаграмму

1. На панели инструментов нажмите кнопку Object Link (Связь объекта).
2. Проведите мышью от действующего лица Продавец к объекту Выбор варианта заказа.
3. Повторите этапы 1 и 2, соединив связями следующие объекты:

# Действующее лицо Продавец и объект Детали Заказа.

# Объект Выбор варианта заказа и объект Детали заказа.

# Объект Детали заказа и объект Заказ №1234.

1. На панели инструментов нажмите кнопку Link Message (Сообщение связи).
2. Щелкните на связи между Продавцом и Выбором варианта заказа.
3. Выделив сообщение, введите его имя "Create New Order -- Создать новый заказ".
4. Повторите этапы с 4 по 6, поместив на диаграмму все остальные сообщения, как показано ниже:

# Open form -- Открыть форму (между Выбором варианта заказа и Деталями заказа)

# Enter order number, customer, order items -- Ввести номер заказа, заказчика и число заказываемых предметов (между Продавцом и Деталями заказа)

# Save the order -- Сохранить заказ (между Продавцом и Деталями заказа)

# Create new, blank order -- Создать пустой заказ (между Деталями заказа и Заказом №1234)

# Set the order number, customer, order items -- Ввести номер заказа, заказчика и число заказываемых предметов (между Деталями заказа и Заказом №1234).

# Save the order -- Сохранить заказ (между Деталями заказа и Заказом №1234)

Теперь, как и раньше, надо продолжить работу и поместить на диаграмму дополнительные элементы, а также рассмотреть ответственности объектов.

### Добавление на диаграмму дополнительных объектов

1. На панели инструментов нажмите кнопку Object.
2. Щелкните мышью где-нибудь на диаграмме, чтобы поместить туда новый объект.
3. Введите имя объекта - Order Manager (Управляющий заказами).
4. На панели инструментов нажмите кнопку Object.
5. Поместите на диаграмму еще один объект.
6. Введите его имя - Transaction Manager (Управляющий транзакциями).

### Назначение ответственностей объектам

1. Выделите сообщение 5 (Создать пустой заказ). Выделяйте слова, а не стрелку.
2. Нажмите комбинацию клавиш CTRL + D, чтобы удалить это сообщение.
3. Повторите этапы 1 и 2, чтобы удалить сообщения 6 и 7:

# Вести номер заказа, заказчика и число заказываемых предметов

# Сохранить заказ

1. Выделите связь между объектами Детали заказа и Заказ №1234.
2. Нажмите комбинацию клавиш CTRL + D, чтобы удалить эту связь.
3. На панели инструментов нажмите кнопку Object Link (Связь объекта).
4. Нарисуйте связь между Деталями Заказа и Управляющим заказами.
5. На панели инструментов нажмите кнопку Object Link (Связь объекта).
6. Нарисуйте связь между Управляющим заказами и Заказом №1234.
7. На панели инструментов нажмите кнопку Object Link (Связь объекта).
8. Нарисуйте связь между Заказом №1234 и Управляющим транзакций.
9. На панели инструментов нажмите кнопку Object Link (Связь объекта).
10. Нарисуйте связь между Управляющим заказами и Управляющим транзакций.
11. На панели инструментов нажмите кнопку Link Message (Сообщение связи).
12. Щелкните на связи между объектами Детали заказа и Управляющим заказами, чтобы ввести новое сообщение.
13. Назовите это сообщение Save the order (Сохранить заказ).
14. Повторите этапы 14 - 16, добавив сообщения с шестого по девятое и назвав их:

# Create new, blank order (Создать новый заказ) - между Управляющим заказами и Заказом №1234.

# Set the order number, customer, order items (Вести номер заказа, заказчика и число заказываемых предметов) - между Управляющим заказами и Заказом №1234.

# Save the order (Сохранить заказ) - между Управляющим заказами и Управляющим транзакциями.

# Collect order information (Информация о заказе) - между Управляющим транзакциями и Заказом №1234.

1. На панели инструментов нажмите кнопку Message to Self (Сообщение себе).
2. Щелкните на объекте Управляющий транзакциями, добавив к нему рефлексивное сообщение.
3. На панели инструментов нажмите кнопку Link Message (Сообщение связи).
4. Щелкните на рефлексивной связи Управляющего транзакциями, чтобы ввести туда сообщение.
5. Назовите новое сообщение Save the order information to the database (Сохранить информацию о заказе в базе данных).

### Соотнесение объектов с классами

1. Найдите в броузере класс OrderOptions.
2. Перетащите его на объект Выбор варианта заказа на диаграмме.
3. Повторите этапы 1 и 2, соотнеся остальные объекты и соответствующие им классы:

# Класс OrderDetail соотнесите с объектом Детали заказа.

# Класс OrderMgr - с объектом Управляющий заказами.

# Класс Order - с объектом Заказ №1234.

# Класс TransactionMgr - с объектом Управляющий транзакциями.

### Соотнесение объектов с классами

1. Щелкните правой кнопкой мыши на объекте Выбор варианта заказа.
2. В открывшемся меню выберите пункт Open Specification (Открыть спецификацию).
3. В раскрывающемся списке классов выберите пункт <New> (Создать). Появится окно спецификации классов.
4. В поле имени введите имя OrderOptions (Выбор заказа).
5. Щелкните на кнопке ОК. Вы вернетесь к окну спецификации объекта.
6. В списке классов выберите теперь класс OrderOptions.
7. Щелкните на кнопке ОК, чтобы вернуться к диаграмме. Теперь объект называется Order Options Form : OrderOptions (Выбор варианта заказа : OrderOptions).
8. Для соотнесения остальных объектов с классами повторите этапы с 1 по 7:

# Класс OrderDetail соотнесите с объектом Детали заказа.

# Класс OrderMgr - с объектом Управляющий заказами.

# Класс Order - с объектом Заказ №1234.

# Класс TransactionMgr - с объектом Управляющий транзакциями.

### Соотнесение сообщений с операциями

1. Щелкните правой кнопкой на сообщении 1, Создать новый заказ.
2. В открывшемся меню выберите пункт Open Specification (Открыть спецификацию).
3. В раскрывающемся списке имен укажите имя операции - Create (Создать).
4. Нажмите на кнопку ОК.
5. Повторите этапы с первого по четвертый для соотнесения с операциями остальных сообщений:

# Сообщение 2:Открыть соотнесите с операцией Open()

# Сообщение 3: Ввести номер заказа, заказчика и число заказываемых предметов - с операцией SubmitInfo().

# Сообщение 4:Сохранить заказ - с операцией Save().

# Сообщение 5:Сохранить заказ - с операцией SaveOrder().

# Сообщение 6:Создать пустой заказ - с операцией Create().

# Сообщение 7: Ввести номер заказа, заказчика и число заказываемых предметов - с операцией SetInfo().

# Сообщение 8:Сохранить заказ - с операцией SaveOrder().

# Сообщение 9:Информация о заказе - с операцией GetInfo().

# Сообщение 10:Сохранить информацию о заказе в базе данных - с операцией Commit().

### Соотнесение сообщений с операциями

1. Щелкните правой кнопкой на сообщении 1, Создать новый заказ.
2. В открывшемся меню выберите пункт <new operation> (создать операцию). Появится окно спецификации операции.
3. В поле имени введите имя операции - Create (Создать).
4. Нажмите на кнопку ОК, чтобы закрыть окно спецификации операции и вернуться на диаграмму.
5. Еще раз щелкните правой кнопкой мыши на сообщении 1.
6. В открывшемся меню выберите пункт Open Specification (Открыть спецификацию).
7. В раскрывающемся списке Name (имя) укажите имя новой операции.
8. Нажмите на кнопку ОК.
9. Повторите этапы с первого по восьмой, чтобы создать новые операции и соотнести с ними остальные сообщения:

# Сообщение 2:Открыть соотнесите с операцией Open()

# Сообщение 3: Ввести номер заказа, заказчика и число заказываемых предметов - с операцией SubmitInfo().

# Сообщение 4:Сохранить заказ - с операцией Save().

# Сообщение 5:Сохранить заказ - с операцией SaveOrder().

# Сообщение 6:Создать пустой заказ - с операцией Create().

# Сообщение 7: Ввести номер заказа, заказчика и число заказываемых предметов - с операцией SetInfo().

# Сообщение 8:Сохранить заказ - с операцией SaveOrder().

# Сообщение 9:Информация о заказе - с операцией GetInfo().

# Сообщение 10:Сохранить информацию о заказе в базе данных - с операцией Commit.

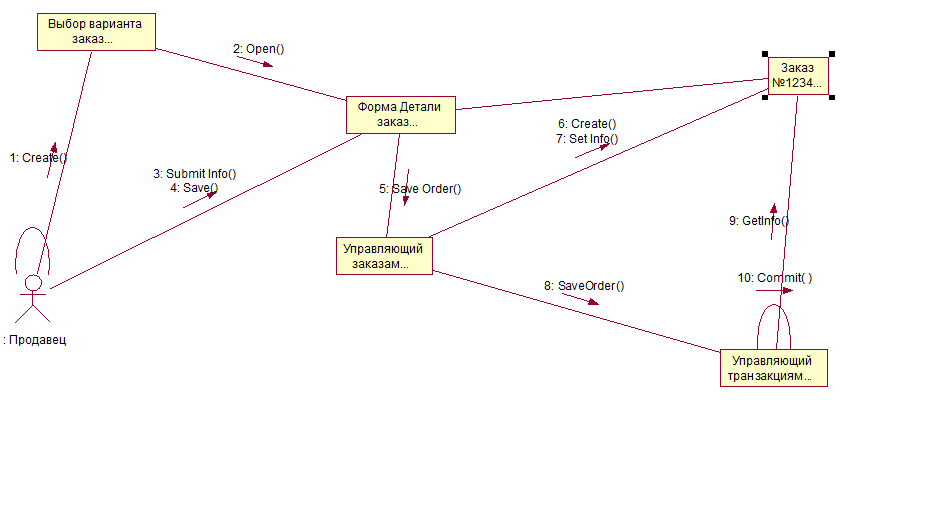


Рис 4.Кооперативная диаграмма с показанными на ней операциями.

# **Лабораторная работа №3. Создание диаграмм классов**

***Цель работы:***

* изучение диаграмм классов,
* изучение их применения в процессе проектирования.

***Теория:***

Центральное место в ООАП занимает разработка логической модели системы в виде диаграммы классов. Нотация классов в языке UML проста и интуитивно понятна всем, кто когда-либо имел опыт работы с CASE-инструментариями. Схожая нотация применяется и для объектов - экземпляров класса, с тем различием, что к имени класса добавляется имя объекта и вся надпись подчеркивается.

Нотация UML предоставляет широкие возможности для отображения дополнительной информации (абстрактные операции и классы, стереотипы, общие и частные методы, детализированные интерфейсы, параметризованные классы). При этом возможно использование графических изображений для ассоциаций и их специфических свойств, таких как отношение агрегации, когда составными частями класса могут выступать другие классы.

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.

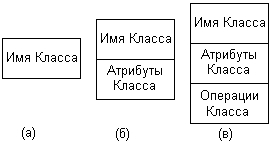
Диаграмма классов представляет собой некоторый граф, вершинами которого являются элементы типа "классификатор", которые связаны различными типами структурных отношений. Следует заметить, что диаграмма классов может также содержать интерфейсы, пакеты, отношения и даже отдельные экземпляры, такие как объекты и связи. Когда говорят о данной диаграмме, имеют в виду статическую структурную модель проектируемой системы. Поэтому диаграмму классов принято считать графическим представленном таких структурных взаимосвязей логической модели системы, которые не зависят или инвариантны от времени.

Диаграмма классов состоит из множества элементов, которые в совокупности отражают декларативные знания о предметной области. Эти знания интерпретируются в базовых понятиях языка UML, таких как классы, интерфейсы и отношения между ними и их составляющими компонентами. При этом отдельные компоненты этой диаграммы могут образовывать пакеты для представления более общей модели системы. Если диаграмма классов является частью некоторого пакета, то ее компоненты должны соответствовать элементам этого пакета, включая возможные ссылки на элементы из других пакетов.

В общем случае пакет статической структурной модели может быть представлен в виде одной или нескольких диаграмм классов. Декомпозиция некоторого представления на отдельные диаграммы выполняется с целью удобства и графической визуализации структурных взаимосвязей предметной области. При этом компоненты диаграммы соответствуют элементам статической семантической модели. Модель системы, в свою очередь, должна быть согласована с внутренней структурой классов, которая описывается на языке UML.

5.1. Класс

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 (рис. 5.1). В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы).

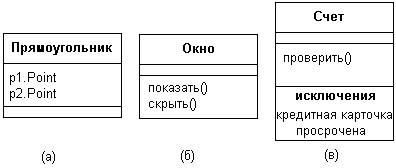


**Рис. 5.1.** Графическое изображение класса на диаграмме классов

Обязательным элементов обозначения класса является его имя. На начальных этапах разработки диаграммы отдельные классы могут обозначаться простым прямоугольником с указанием только имени соответствующего класса (рис. 5.1, а). По мере проработки отдельных компонентов диаграммы описания классов дополняются атрибутами (рис. 5.1, б) и операциями (рис. 5.1, в).

Предполагается, что окончательный вариант диаграммы содержит наиболее полное описание классов, которые состоят из трех разделов или секций. Иногда в обозначениях классов используется дополнительный четвертый раздел, в котором приводится семантическая информация справочного характера или явно указываются исключительные ситуации.

Даже если секция атрибутов и операций является пустой, в обозначении класса она выделяется горизонтальной линией, чтобы сразу отличить класс от других элементов языка UML. Примеры графического изображения классов на диаграмме классов приведены на рис. 5.2. В первом случае для класса "Прямоугольник" (рис. 5.2, а) указаны только его атрибуты - точки на координатной плоскости, которые определяют его расположение. Для класса "Окно" (рис. 5.2, б) указаны только его операции, секция атрибутов оставлена пустой. Для класса "Счет" (рис. 5.2, в) дополнительно изображена четвертая секция, в которой указано исключение - отказ от обработки просроченной кредитной карточки.



**Рис.5.2.** Примеры графического изображения классов на диаграмме

Имя класса

Имя класса должно быть уникальным в пределах пакета, который описывается некоторой совокупностью диаграмм классов (возможно, одной диаграммой). Оно указывается в первой верхней секции прямоугольника. В дополнение к общему правилу наименования элементов языка UML, имя класса записывается по центру секции имени полужирным шрифтом и должно начинаться с заглавной буквы. Рекомендуется в качестве имен классов использовать существительные, записанные по практическим соображениям без пробелов. Необходимо помнить, что именно имена классов образуют словарь предметной области при ООАП.

В первой секции обозначения класса могут находиться ссылки на стандартные шаблоны или абстрактные классы, от которых образован данный класс и, соответственно, от которых он наследует свойства и методы. В этой секции может приводиться информация о разработчике данного класса и статус состояния разработки, а также могут записываться и другие общие свойства этого класса, имеющие отношение к другим классам диаграммы или стандартным элементам языка UML.

Примерами имен классов могут быть такие существительные, как "Сотрудник", "Компания", "Руководитель", "Клиент", "Продавец", "Менеджер", "Офис" и многие другие, имеющие непосредственное отношение к моделируемой предметной области и функциональному назначению проектируемой системы.

Класс может не иметь экземпляров или объектов. В этом случае он называется абстрактным классом, а для обозначения его имени используется наклонный шрифт (курсив). В языке UML принято общее соглашение о том, что любой текст, относящийся к абстрактному элементу, записывается курсивом. Данное обстоятельство является семантическим аспектом описания соответствующих элементов языка UML.

**Примечание**

В некоторых случаях необходимо явно указать, к какому пакету относится тот или иной класс. Для этой цели используется специальный символ разделитель - двойное двоеточие "::". Синтаксис строки имени класса в этом случае будет следующий <Имя\_пакета>::<Имя\_класса>. Другими словами, перед именем класса должно быть явно указано имя пакета, к которому его следует отнести. Например, если определен пакет с именем "Банк", то класс "Счет" в этом банке может быть записан в виде: "Банк::Счет".

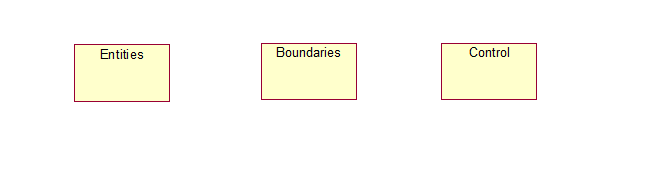
***Упражнения:***

### Настройка

1. В меню модели выберите пункт Tools > Options (Инструменты > Параметры).
2. Перейдите на вкладку диаграмм.
3. Убедитесь, что помечен контрольный переключатель Show Stereotypes (Показать стереотипы).
4. Убедитесь, что помечены контрольные переключатели Show All Attributes (Показать все атрибуты) и Show All Operations (Показать все операции).
5. Убедитесь, что не помечены переключатели Suppress Attributes (Подавить вывод атрибутов) и Suppress Operations (Подавить вывод операций).

### Создание пакетов

1. Щелкните правой кнопкой мыши на Логическом представлении броузера.
2. В открывшемся меню выберите пункт New > Package (Создать > пакет).
3. Назовите новый пакет Entities (Сущности).
4. Повторите этапы с первого по третий, создав пакеты Boundaries (границы) и Control (управление).

 Рис 5.Главная диаграмма Классов системы обработки заказов.

### Создание Главной диаграммы Классов

1. Дважды щелкните на Главной диаграмме Классов прямо под Логическим представлением броузера, чтобы открыть ее.
2. Перетащите пакет Entities из броузера на диаграмму.
3. Перетащите пакеты Boundaries и Control из броузера на диаграмму.

### Создание диаграммы Классов для сценария "Ввести новый заказ" со всеми классами.

1. Щелкните правой кнопкой мыши на Логическом представлении броузера.

2. В открывшемся меню выберите пункт New > Class Diagram (Создать > Диаграмму Классов).

3. Назовите новую диаграмму Классов Add New Order (Введение нового заказа).

4. Щелкните в броузере на этой диаграмме дважды, чтобы открыть ее.

5. Перетащите из броузера все классы (OrderOptions, OrderDetail, Order, OrderMgr и TransactionMgr).

### Добавление стереотипов к классам

1. Щелкните правой кнопкой мыши на классе OrderOptions диаграммы.
2. В открывшемся меню выберите пункт Open Specification (Открыть спецификацию).
3. В поле стереотипа введите слово Boundary.
4. Нажмите на кнопку ОК.
5. Щелкните правой кнопкой мыши на классе OrderDetail диаграммы.
6. В открывшемся меню выберите пункт Open Specification (Открыть спецификацию).
7. В раскрывающемся списке в поле стереотипов теперь будет стереотип Boundary. Укажите его.
8. Нажмите на кнопку ОК.
9. Повторите этапы 1 - 4, связав классы OrderMgr и TransactionMgr со стереотипом Control, а класс Order - со стереотипом Entity.

### Объединение классов в пакеты

1. Перетащите в броузере класс OrderOptions на пакет Boundaries.

2. Перетащите класс OrderDetail на пакет Boundaries.

3. Перетащите классы OrderMgr и TransactionMgr на пакет Control.

4. Перетащите класс Order на пакет Entities.

### Добавление диаграмм Классов к каждому пакету

1. Щелкните правой кнопкой на пакете Boundaries броузера.
2. В открывшемся меню выберите пункт New > Class Diagram (Создать > Диаграмму Классов).
3. Введите имя новой диаграммы - Main (Главная).
4. Дважды щелкните мышью на этой диаграмме, чтобы открыть ее.
5. Перетащите на нее из броузера классы OrderOptions и OrderDetail.
6. Закройте диаграмму.
7. Щелкните правой кнопкой на пакете Entities броузера.
8. В открывшемся меню выберите пункт New > Class Diagram (Создать > Диаграмму Классов).
9. Введите имя новой диаграммы - Main (Главная).
10. Дважды щелкните мышью на этой диаграмме, чтобы открыть ее.
11. Перетащите на нее из броузера класс Order.
12. Закройте диаграмму.
13. Щелкните правой кнопкой на пакете Control броузера.
14. В открывшемся меню выберите пункт New > Class Diagram (Создать >
15. Диаграмму Классов).
16. Введите имя новой диаграммы - Main (Главная).
17. Дважды щелкните мышью на этой диаграмме, чтобы открыть ее.
18. Перетащите на нее из броузера классы OrderMgr и TransactionMgr.
19. Закройте диаграмму.

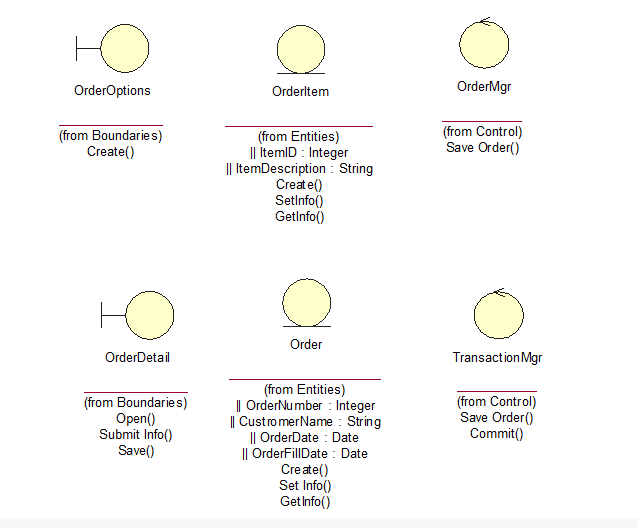


Рис 6.Стереотипы классов для варианта использования Ввести новый заказ.

# **Лабораторная работа №4. Создание диаграмм классов (учет новых требований)**

***Цель работы*:**

* изучение диаграмм классов,
* изучение их применения в процессе проектирования.

***Теория:***

**Атрибуты класса**

Во второй сверху секции прямоугольника класса записываются его атрибуты (attributes) или свойства. В языке UML принята определенная стандартизация записи атрибутов класса, которая подчиняется некоторым синтаксическим правилам. Каждому атрибуту класса соответствует отдельная строка текста, которая состоит из квантора видимости атрибута, имени атрибута, его кратности, типа значений атрибута и, возможно, его исходного значения:

<квантор видимости><имя атрибута>[кратность]:

<тип атрибута> = <исходное значение>{строка-свойство}

Квантор видимости может принимать одно из трех возможных значений и, соответственно, отображается при помощи специальных символов:

* Символ "+" обозначает атрибут с областью видимости типа общедоступный (public). Атрибут с этой областью видимости доступен или виден из любого другого класса пакета, в котором определена диаграмма.
* Символ "#" обозначает атрибут с областью видимости типа защищенный (protected). Атрибут с этой областью видимости недоступен или невиден для всех классов, за исключением подклассов данного класса.
* И, наконец, знак "-" обозначает атрибут с областью видимости типа закрытый (private). Атрибут с этой областью видимости недоступен или невиден для всех классов без исключения.

Квантор видимости может быть опущен. В этом случае его отсутствие просто означает, что видимость атрибута не указывается. Эта ситуация отличается от принятых по умолчанию соглашений в традиционных языках программирования, когда отсутствие квантора видимости трактуется как public или private. Однако вместо условных графических обозначений можно записывать соответствующее ключевое слово: public, protected, private.

**Примечание**

Поскольку язык UML инвариантен относительно реализации своих конструкций в конкретных языках программирования, семантика отдельных кванторов видимости не является строго фиксированной. Значения этих кванторов должны дополнительно уточняться пояснительным текстом на естественном языке или соглашением по использованию соответствующих программно-зависимых синтаксических конструкций.

Имя атрибута представляет собой строку текста, которая используется в качестве идентификатора соответствующего атрибута и поэтому должна быть уникальной в пределах данного класса. Имя атрибута является единственным обязательным элементом синтаксического обозначения атрибута.

Кратность атрибута характеризует общее количество конкретных атрибутов данного типа, входящих в состав отдельного класса. В общем случае кратность записывается в форме строки текста в квадратных скобках после имени соответствующего атрибута:

[нижняя\_граница1 .. верхняя\_граница1, нижняя\_граница2.. верхняя\_грашца2, ..., нuжняя\_гpaнuцak .. верхняя\_границаk],

где нижняя\_граница и верхняя\_граница являются положительными целыми числами, каждая пара которых служит для обозначения отдельного замкнутого интервала целых чисел, у которого нижняя (верхняя) граница равна значению нижняя\_граница (верхняя\_граница). В целом данное условное обозначение кратности соответствует теоретико-множественному объединению соответствующих интервалов. В качестве верхней\_границы может использоваться специальный символ "\*", который означает произвольное положительное целое число. Другими словами, это означает неограниченное сверху значение кратности соответствующего атрибута.

Значения кратности из интервала следуют в монотонно возрастающем порядке без пропуска отдельных чисел, лежащих между нижней и верхней границами. При этом придерживаются следующего правила: соответствующие нижние и верхние границы интервалов включаются в значение кратности. Если в качестве кратности указывается единственное число, то кратность атрибута принимается равной данному числу. Если же указывается единственный знак "\*", то это означает, что кратность атрибута может быть произвольным положительным целым числом или нулем.

В качестве примера рассмотрим следующие варианты задания кратности атрибутов.

* [0..1] означает, что кратность атрибута может принимать значение О или 1. При этом 0 означает отсутствие значения для данного атрибута.
* [0..\*] означает, что кратность атрибута может принимать любое положительное целое значение большее или равное 0. Эта кратность может быть записана короче в виде простого символа - [\*].
* [1.:\*] означает, что кратность атрибута может принимать любое положительное целое значение большее или равное 1.
* [1..5] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 4, 5.
* [1..3,5,7] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 5, 7.
* [1..3,7.. 10] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 7, 8, 9, 10.
* [1..3,7..\*] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, а также любое положительное целое значение большее или равное 7.

Если кратность атрибута не указана, то по умолчанию принимается ее значение равное 1..1, т. е. в точности 1.

Тип атрибута представляет собой выражение, семантика которого определяется языком спецификации соответствующей модели. В нотации UML тип атрибута иногда определяется в зависимости от языка программирования, который предполагается использовать для реализации данной модели. В простейшем случае тип атрибута указывается строкой текста, имеющей осмысленное значение в пределах пакета или модели, к которым относится рассматриваемый класс.

Можно привести следующие примеры задания имен и типов атрибутов классов:

* цвет: Соlоr - здесь цвет является именем атрибута, Color - именем типа данного атрибута. Указанная запись может определять традиционно используемую RGB-модель (красный, зеленый, синий) для представления цвета. В этом случае имя типа Color как раз и характеризует семантическую конструкцию, которая применяется в большинстве языков программирования для представления цвета.
* имя\_сотрудника [1..2] : String - здесь имя\_сотрудника является именем атрибута, который служит для представления информации об имени, а возможно, и отчестве конкретного сотрудника. Тип атрибута String (Строка) как раз и указывает на тот факт, что отдельное значение имени представляет собой строку текста из одного или двух слов (например, "Кирилл" или "Дмитрий Иванович"). Поскольку во многих языках программирования существует тип данных String, использование соответствующего англоязычного термина не вызывает недоразумения у большинства программистов. Однако, хотя в языке UML все термины даются в англоязычном представлении, использование в качестве типа атрибута Строка в данной ситуации не исключается и определяется только соображениями удобства.
* видимость:Boolean - здесь видимость есть имя абстрактного атрибута (курсив здесь не случаен), который может характеризовать наличие визуального представления соответствующего класса на экране монитора. В этом случае тип Boolean означает, что возможными значениями данного атрибута является одно из двух логических значений: истина (true) или ложь (false). При этом значение истина может соответствовать наличию графического изображения на экране монитора, а значение ложь - его отсутствию, о чем дополнительно указывается в пояснительном тексте. Поскольку кратность атрибута видимость не указана, она принимает значение 1 по умолчанию. В этой ситуации англоязычное имя типа атрибута вполне оправдано наличием соответствующего базового типа в языках программирования. Абстрактный характер данного атрибута обозначается курсивным текстом в записи данного атрибута.
* форма:Многоугольник - здесь имя атрибута форма может характеризовать такой класс, который является геометрической фигурой на плоскости. В этом случае тип атрибута Многоугольник указывает на тот факт, что отдельная геометрическая фигура может иметь форму треугольника, прямоугольника, ромба, пятиугольника и любого другого многоугольника, но не окружности или эллипса. Вполне очевидно, что в данной ситуации использование соответствующего англоязычного термина вряд ли целесообразно, поскольку тип Многоугольник не является базовым для языков программирования.

Исходное значение служит для задания некоторого начального значения для соответствующего атрибута в момент создания отдельного экземпляра класса. Здесь необходимо придерживаться правила принадлежности значения типу конкретного атрибута. Если исходное значение не указано, то значение соответствующего атрибута не определено на момент создания нового экземпляра класса. С другой стороны, конструктор соответствующего объекта может переопределять исходное значение в процессе выполнения программы, если в этом возникает необходимость.

В качестве примеров исходных значений атрибутов можно привести следующие дополненные выше варианты задания атрибутов:

* цвет:Соlоr = (255, 0, 0) - в RGB-модели цвета это соответствует чистому красному цвету в качестве исходного значения для данного атрибута.
* имя\_сотрудника[1..2]:String = Иван Иванович - возможно, это нетипичный случай, который, скорее, соответствует ситуации имя\_руководителя[2]:81пп§ = Иван Иванович.
* видимость:Вооlеаn = истина - может соответствовать ситуации, когда в момент создания экземпляра класса создается видимое на экране монитора окно, соответствующее данному объекту.
* форма:Многоугольник = прямоугольник - вряд ли требует комментариев, поскольку здесь речь идет о геометрической форме создаваемого объекта.

При задании атрибутов могут быть использованы две дополнительные синтаксические конструкции - это подчеркивание строки атрибута и пояснительный текст в фигурных скобках.

Подчеркивание строки атрибута означает, что соответствующий атрибут может принимать подмножество значений из некоторой области значений атрибута, определяемой его типом. Эти значения можно рассматривать как набор однотипных записей или массив, которые в совокупности характеризуют каждый объект класса.

Например, если некоторый атрибут задан в виде форма: Прямоугольник. то это будет означать, что все объекты данного класса могут иметь несколько различных форм, каждая из которых является прямоугольником. Другим примером может служить задание атрибута в виде номер\_счета:Integer. что может означать для объекта Сотрудник наличие некоторого подмножества счетов, общее количество которых заранее не фиксируется.

Строка-свойство служит для указания значений атрибута, которые не могут быть изменены в программе при работе с данным типом объектов. Фигурные скобки как раз и обозначают фиксированное значение соответствующего атрибута для класса в целом, которое должны принимать все вновь создаваемые экземпляры класса без исключения. Это значение принимается за исходное значение атрибута, которое не может быть переопределено в последующем. Отсутствие строки-свойства по умолчанию трактуется так, что значение соответствующего атрибута может быть изменено в программе. Например, строка-свойство в записи атрибута заработная\_плата:Currency = = {$500} может служить для обозначения фиксированной заработной платы для каждого объекта класса "Сотрудник" определенной должности в некоторой организации. С другой стороны, запись данного атрибута в виде зара-ботная\_плата: Currency = $500 означает уже нечто иное, а именно - при создании нового экземпляра Сотрудник (аналогия - прием на работу нового сотрудника) для него устанавливается по умолчанию заработная плата в $500. Однако для отдельных сотрудников могут быть сделаны исключения как в большую, так и в меньшую сторону, о чем необходимо позаботиться дополнительно в программе.

***Упражнения:***

### Настройка

1. В меню модели выберите пункт Tools > Options.
2. Перейдите на вкладку Diagram.
3. Убедитесь, что переключатель Show Visibility помечен.
4. Убедитесь, что переключатель Show Stereotypes помечен.
5. Убедитесь, что переключатель Show Operation Signatures помечен.
6. Убедитесь, что переключатели Show All Attributes и Show All Operations помечены.
7. Убедитесь, что переключатели Suppress Attributes и Suppress Operations не помечены.
8. Перейдите на вкладку Notation.
9. Убедитесь, что переключатель Visibility as Icons не помечен.

### Добавление нового класса

1. Найдите в броузере диаграмму Классов варианта использования "Ввести новый заказ".
2. Щелкните на ней дважды, чтобы ее открыть.
3. Нажмите кнопку Class панели инструментов.
4. Щелкните мышью внутри диаграммы, чтобы поместить там новый класс.
5. Назовите его OrderItem (ПозицияЗаказа).
6. Назначьте этому классу стереотип Entity.
7. В броузере перетащите класс в пакет Entities.

### Добавление атрибутов

1. Щелкните правой кнопкой мыши на классе Order (Заказ).
2. В открывшемся меню выберите пункт New Attribute (Создать атрибут).
3. Введите новый атрибут OrderNumber : Integer (НомерЗаказа)
4. Нажмите клавишу Enter.
5. Введите следующий атрибут CustomerName : String (НаименованиеЗаказчика).
6. Повторите этапы 4 и 5, добавив атрибуты OrderDate : Date (ДатаЗаказа) и OrderFillDate : Date (ДатаЗаполненияЗаказа).
7. Щелкните правой кнопкой мыши на классе OrderItem.
8. В открывшемся меню выберите пункт New Attribute (Создать атрибут).
9. Введите новый атрибут ItemID : Integer (ИдентификаторПредмета).
10. Нажмите клавишу Enter.
11. Введите следующий атрибут ItemDescription : String (ОписаниеПредмета).

### Добавление операций к классу OrderItem

1. Щелкните правой кнопкой мыши на классе OrderItem.
2. В открывшемся меню выберите пункт New Operation (Создать операцию).
3. Введите новую операцию Create.
4. Нажмите клавишу Enter.
5. Введите следующую операцию SetInfo
6. Нажмите клавишу Enter.
7. Введите следующую операцию GetInfo.

### Подробное описание операций с помощью диаграммы Классов

1. Щелкните мышью на классе Order, выделив его таким способом.
2. Щелкните на этом классе еще один раз, чтобы переместить курсор внутрь.
3. Отредактируйте операцию Create(), чтобы она выглядела следующим образом: Create() : Boolean
4. Отредактируйте операцию SetInfo(), чтобы она выглядела следующим образом: SetInfo(OrderNum : Integer, Customer : String, OrderDate : Date, FillDate : Date) : Boolean
5. Отредактируйте операцию GetInfo(), чтобы она выглядела следующим образом: GetInfo() : String

### Подробное описание операций с помощью броузера

1. Найдите в броузере класс OrderItem.
2. Чтобы раскрыть этот класс, щелкните на значке "+" рядом с ним. В броузере появятся его атрибуты и операции.
3. Дважды щелкните на операции GetIInfo(), чтобы открыть окно ее спецификации.
4. В раскрывающемся списке Return class (возвращаемый класс) укажите String.
5. Щелкните на кнопке ОК, закрыв окно спецификации операции.
6. Дважды щелкните в броузере на операции SetInfo класса OrderItem, чтобы открыть окно ее спецификации.
7. В раскрывающемся списке Return class укажите Boolean.
8. Перейдите на вкладку Detail (Подробно).
9. Щелкните правой кнопкой мыши на белом поле в области аргументов, чтобы добавить туда новый параметр.
10. В открывшемся меню выберите пункт Insert. Rose добавит туда аргумент под названием argname.
11. Щелкните один раз на этом слове, чтобы выделить его, и измените имя аргумента на ID.
12. Щелкните на колонке Type, открыв раскрывающийся список типов. В нем выберите тип Integer.
13. Щелкните на колонке Default, чтобы добавить значение аргумента по умолчанию. Введите туда число 0.
14. Нажмите на кнопку ОК, закрыв окно спецификации операции.
15. Дважды щелкните на операции Create() класса OrderItem, чтобы открыть окно ее спецификации.
16. В раскрывающемся списке Return class укажите Boolean.
17. Нажмите на кнопку ОК, закрыв окно спецификации операции.

### Подробное описание операций с помощью любого из описанных методов

1. Используя броузер или диаграмму Классов, введите следующую сигнатуру операций класса OrderDetail:

Open() : Boolean

SubmitInfo() : Boolean

Save() : Boolean

1. Используя броузер или диаграмму Классов, введите следующую сигнатуру операций класса OrderOptions:

Create() : Boolean

1. Используя броузер или диаграмму Классов, введите следующую сигнатуру операций класса OrderMgr:

SaveOrder(OrderID : Integer) : Boolean

1. Используя броузер или диаграмму Классов, введите следующую сигнатуру операций класса TransactionMgr:

SaveOrder(OrderID : Integer) : Boolean

Commit() : Integer

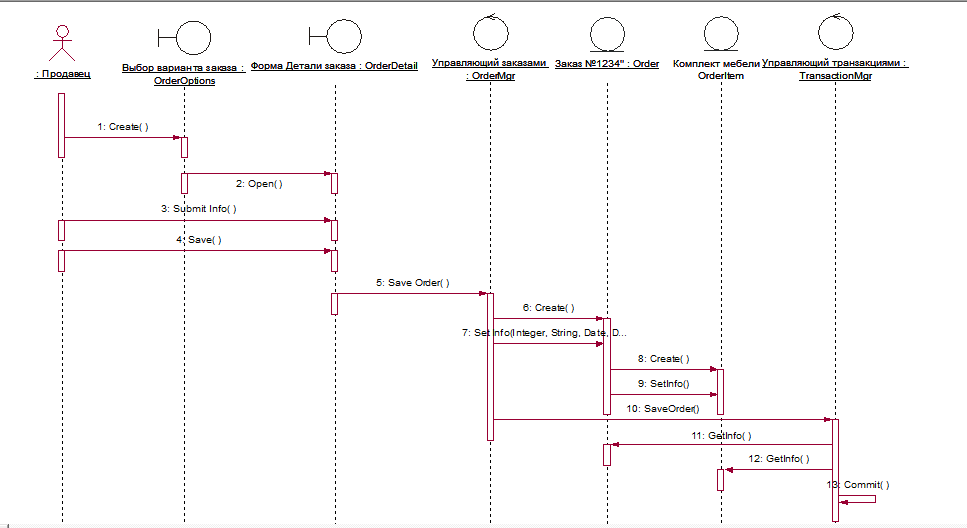


Рис 7. Создание диаграмм классов (учет новых требований)

**Лабораторная работа №5.** [**Создание диаграмм классов (добавление связей между классами)**](file:///C:\Users\NotePad.by\Desktop\%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5%20%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%20%D0%B8%20%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5.%20%D0%A7%D0%B0%D1%81%D1%82%D1%8C%202\files\ood_lb\ex5.doc)

***Цель работы:***

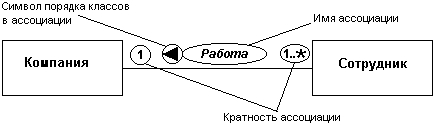
* изучение диаграмм классов,
* изучение их применения в процессе проектирования.

***Теория:***

Отношение ассоциации соответствует наличию некоторого отношения между классами. Данное отношение обозначается сплошной линией с дополнительными специальными символами, которые характеризуют отдельные свойства конкретной ассоциации. В качестве дополнительных специальных символов могут использоваться имя ассоциации, а также имена и кратность классов-ролей ассоциации. Имя ассоциации является необязательным элементом ее обозначения. Если оно задано, то записывается с заглавной (большой) буквы рядом с линией соответствующей ассоциации.

Наиболее простой случай данного отношения - бинарная ассоциация. Она связывает в точности два класса и, как исключение, может связывать класс с самим собой. Для бинарной ассоциации на диаграмме может быть указан порядок следования классов с использованием треугольника в форме стрелки рядом с именем данной ассоциации. Направление этой стрелки указывает на порядок классов, один из которых является первым (со стороны треугольника), а другой - вторым (со стороны вершины треугольника). Отсутствие данной стрелки рядом с именем ассоциации означает, что порядок следования классов в рассматриваемом отношении не определен.

В качестве простого примера отношения бинарной ассоциации рассмотрим отношение между двумя классами - классом "Компания" и классом "Сотрудник" (рис. 5.5). Они связаны между собой бинарной ассоциацией Работа, имя которой указано на рисунке рядом с линией ассоциации. Для данного отношения определен порядок следования классов, первым из которых является класс "Сотрудник", а вторым - класс "Компания". Отдельным примером или экземпляром данного отношения может являться пара значений (Петров И. И., "Рога&Копыта"). Это означает, что сотрудник Петров И. И. работает в компании "Рога&Копыта".



**Рис. 5.5.** Графическое изображение отношения бинарной ассоциации между классами

Тернарная ассоциация и ассоциации более высокой арности в общем случае называются N-арной ассоциацией (читается - "эн арная ассоциация"). Такая ассоциация связывает некоторым отношением 3 и более классов, при этом один класс может участвовать в ассоциации более чем один раз. Класс ассоциации имеет определенную роль в соответствующем отношении, что может быть явно указано на диаграмме. Каждый экземпляр N-арной ассоциации представляет собой N-арный кортеж значений объектов из соответствующих классов. Бинарная ассоциация является частным случаем N-арной ассоциации, когда значение N=2, и имеет свое собственное обозначение.

N-арная ассоциация графически обозначается ромбом, от которого ведут линии к символам классов данной ассоциации. В этом случае ромб соединяется с символами соответствующих классов сплошными линиями. Обычно линии проводятся от вершин ромба или от середины его сторон. Имя N-арной ассоциации записывается рядом с ромбом соответствующей ассоциации.

Порядок классов в N-арной ассоциации, в отличие от порядка множеств в отношении, на диаграмме не фиксируется. Некоторый класс может быть присоединен к ромбу пунктирной линией. Это означает, что данный класс обеспечивает поддержку свойств соответствующей N-арной ассоциации, а сама N-арная ассоциация имеет атрибуты, операции и/или ассоциации. Другими словами, такая ассоциация, в свою очередь, является классом с соответствующим обозначением в виде прямоугольника и является самостоятельным элементом языка UML - ассоциацией-классом (Association Class). N-арная ассоциация не может содержать символ агрегации ни для какой из своих ролей.

В качестве примера конкретной тернарной ассоциации рассмотрим отношение между тремя классами: "Футбольная команда", "Год" и "Игра". Данная ассоциация указывает на наличие отношения между этими тремя классами, которое может представлять информацию об играх футбольных команд в национальном чемпионате в течение нескольких последних лет (рис. 5.6).

Как уже упоминалось, отдельный класс ассоциации имеет собственную роль в отношении. Эта роль может быть изображена графически на диаграмме классов. С этой целью в языке UML вводится в рассмотрение специальный элемент - конец ассоциации (Association End), который графически соответствует точке соединения линии ассоциации с отдельным классом. Конец ассоциации является частью ассоциации, но не класса. Каждая ассоциация имеет два или больше концов ассоциации. Наиболее важные свойства ассоциации указываются на диаграмме рядом с этими элементами ассоциации и должны перемешаться вместе с ними.



**Рис. 5.6.** Графическое изображение тернарной ассоциации между тремя классами

Одним из таких дополнительных обозначений является имя роли отдельного класса, входящего в ассоциацию. Имя роли представляет собой строку текста рядом с концом ассоциации для соответствующего класса. Она указывает специфическую роль, которую играет класс, являющийся концом рассматриваемой ассоциации. Имя роли не является обязательным элементом обозначений и может отсутствовать на диаграмме.

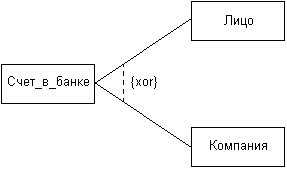
Следующий элемент обозначений - кратность отдельных классов, являющихся концами ассоциации. Кратность отдельного класса обозначается в виде интервала целых чисел, аналогично кратности атрибутов и операций классов. Интервал записывается рядом с концом ассоциации и для N-арной ассоциации означает потенциальное число отдельных экземпляров или значений кортежей этой ассоциации, которые могут иметь место, когда остальные N-1 экземпляров или значений классов фиксированы.

Так, для рассмотренного ранее примера (см. рис. 5.5) кратность "1" для класса "Компания" означает, что каждый сотрудник может работать только в одной компании. Кратность "1..\*" для класса "Сотрудник" означает, что в каждой компании могут работать несколько сотрудников, общее число которых заранее неизвестно и ничем не ограничено. Заметим, что вместо кратности "1..\*" записать только символ "\*" нельзя, поскольку последний означает кратность "0..\*". Для данного примера это означало бы, что отдельные компании могут совсем не иметь сотрудников в своем штате. Но такая кратность вполне приемлема в других ситуациях, как это видно из рассмотренного выше примера (рис. 5.6).

Что касается других свойств отношения, ассоциации, то в случае их наличия, они могут рассматриваться в качестве атрибутов класса ассоциации и могут быть указаны на диаграмме обычным для класса способом в соответствующей секции прямоугольника класса.

Частным случаем отношения ассоциации является так называемая исключающая ассоциация (Xor-association). Семантика данной ассоциации указывает на тот факт, что из нескольких потенциально возможных вариантов данной ассоциации в каждый момент времени может использоваться только один ее экземпляр. На диаграмме классов исключающая ассоциация изображается пунктирной линией, соединяющей две и более ассоциации, рядом с которой записывается строка-ограничение "{хог}".

Например, счет в банке может быть открыт для клиента, в качестве которого может выступать физическое лицо (индивидум) или компания, что изображается с помощью исключающей ассоциации (рис. 5.7).



**Рис. 5.7.** Графическое изображение исключающей ассоциации между тремя классами

Специальной формой или частным случаем отношения ассоциации является отношение агрегации, которое, в свою очередь, тоже имеет специальную форму - отношение композиции. Поскольку эти отношения имеют свои специальные обозначения и относятся к базовым понятиям языка UML, рассмотрим их последовательно.

***Упражнения***

### Настройка

1. Найдите в броузере диаграмму Классов "Ввод нового заказа"
2. Дважды щелкните на ней, чтобы открыть ее.
3. Проверьте, имеется ли на панели инструментов диаграммы кнопка Unidirectional Association. Если ее нет, продолжайте настройку, выполнив этапы 4 и 5. Если есть, приступайте к выполнению самого упражнения.
4. Щелкните правой кнопкой мыши на панели инструментов диаграммы и в открывшемся меню выберите пункт Customize.
5. Добавьте на панель кнопку, называющуюся Create A Unidirectional Association.

### 

Рис 8. [Создание диаграмм классов (добавление связей между классами)](file:///C:\Users\NotePad.by\Desktop\%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5%20%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%20%D0%B8%20%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5.%20%D0%A7%D0%B0%D1%81%D1%82%D1%8C%202\files\ood_lb\ex5.doc)

### Добавление ассоциаций

1. Нажмите кнопку панели инструментов Unidirectional Association.
2. Нарисуйте ассоциацию от класса ВыборЗаказа (OrderOptions) к классу ДеталиЗаказа (OrderDetail).
3. Повторите этапы 1 и 2, создав еще ассоциации:

# От класса OrderDetail к классу МенеджерЗаказов (OrderMgr)

# От класса OrderMgr к классу Заказ (Order)

# От класса OrderMgr к классу МенеджерТранзакций (TransactionMgr)

# От класса TransactionMgr к классу Order

# От класса TransactionMgr к классу ПозицияЗаказа (OrderItem)

# От класса Order к классу OrderItem

1. Щелкните правой кнопкой мыши на однонаправленной ассоциации между классами OrderOptions и OrderDetail, со стороны класса OrderOptions.
2. В открывшемся меню выберите пункт Multiplicity > Zero or One.
3. Щелкните правой кнопкой мыши на другом конце однонаправленной ассоциации.
4. В открывшемся меню выберите пункт Multiplicity > Zero or One.
5. Повторите этапы 4 - 7, добавив на диаграмму значения множественности для остальных ассоциаций

# **Лабораторная работа №6. Создание диаграммы состояний**

***Цель работы:***

* изучение диаграмм состояний,
* изучение их применения в процессе проектирования.

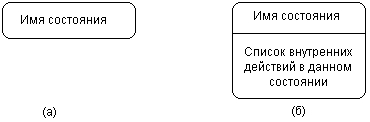
***Теория:***

Понятие состояния (state) является фундаментальным не только в метамоде-ли языка UML, но и в прикладном системном анализе. Ранее в главе 1 кратко были рассмотрены особенности представления динамических характеристик сложных систем, традиционно используемых для моделирования поведения. Вся концепция динамической системы основывается на понятии состояния системы. Однако семантика состояния в языке UML имеет целый ряд специфических особенностей.

В языке UML под состоянием понимается абстрактный метакласс, используемый для моделирования отдельной ситуации, в течение которой имеет место выполнение некоторого условия. Состояние может быть задано в виде набора конкретных значений атрибутов класса или объекта, при этом изменение их отдельных значений будет отражать изменение состояния моделируемого класса или объекта.

Следует заметить, что не каждый атрибут класса может характеризовать его состояние. Как правило, имеют значение только такие свойства элементов системы, которые отражают динамический или функциональный аспект ее поведения. В этом случае состояние будет характеризоваться некоторым инвариантным условием, включающим в себя только значимые для поведения класса атрибуты и их значения.

Например, инвариант может представлять статическую ситуацию, когда объект находится в состоянии ожидания возникновения некоторого внешнего события. С другой стороны, инвариант используется для моделирования динамических аспектов, когда в ходе процесса выполняются некоторые действия. В этом случае моделируемый элемент переходит в рассматриваемое состояние в момент начала соответствующей деятельности и покидает данное состояние в момент ее завершения.



**Рис. 6.2.** Графическое изображение состояний на диаграмме состояний

Состояние на диаграмме изображается прямоугольником со скругленными вершинами (рис. 6.2). Этот прямоугольник, в свою очередь, может быть разделен на две секции горизонтальной линией. Если указана лишь одна секция, то в ней записывается только имя состояния (рис. 6.2, а). В противном случае в первой из них записывается имя состояния, а во второй - список некоторых внутренних действий или переходов в данном состоянии (рис. 6.2, б). При этом под действием в языке UML понимают некоторую атомарную операцию, выполнение которой приводит к изменению состояния или возврату некоторого значения (например, "истина" или "ложь").

  Имя состояния

Имя состояния представляет собой строку текста, которая раскрывает содержательный смысл данного состояния. Имя всегда записывается с заглавной буквы. Поскольку состояние системы является составной частью процесса ее функционирования, рекомендуется в качестве имени использовать глаголы в настоящем времени (звенит, печатает, ожидает) или соответствующие причастия (занят, свободен, передано, получено). Имя у состояния может отсутствовать, т. е. оно является необязательным для некоторых состояний. В этом случае состояние является анонимным, и если на диаграмме состояний их несколько, то все они должны различаться между собой.

  Список внутренних действий

Эта секция содержит перечень внутренних действий или деятельностей, которые выполняются в процессе нахождения моделируемого элемента в данном состоянии. Каждое из действий записывается в виде отдельной строки и имеет следующий формат:

<метка-дёйствия '/' выражение-действия>

Метка действия указывает на обстоятельства или условия, при которых будет выполняться деятельность, определенная выражением действия. При этом выражение действия может использовать любые атрибуты и связи, которые принадлежат области имен или контексту моделируемого объекта. Если список выражений действия пустой, то разделитель в виде наклонной черты '/' может не указываться.

Перечень меток действия имеет фиксированные значения в языке UML, которые не могут быть использованы в качестве имен событий. Эти значения следующие:

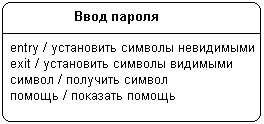
* entry - эта метка указывает на действие, специфицированное следующим за ней выражением действия, которое выполняется в момент входа в данное состояние (входное действие);
* exit - эта метка указывает на действие, специфицированное следующим за ней выражением действия, которое выполняется в момент выхода из данного состояния (выходное действие);
* do - эта метка специфицирует выполняющуюся деятельность ("do activity"), которая выполняется в течение всего времени, пока объект находится в данном состоянии, или до тех пор, пока не закончится вычисление, специфицированное следующим за ней выражением действия.

В последнем случае при завершении события генерируется соответствующий результат;

* include - эта метка используется для обращения к подавтомату, при этом следующее за ней выражение действия содержит имя этого подавтомата.

Во всех остальных случаях метка действия идентифицирует событие, которое запускает соответствующее выражение действия. Эти события называются внутренними переходами и семантически эквивалентны переходам в само это состояние, за исключением той особенности, что выход из этого состояния или повторный вход в него не происходит. Это означает, что действия входа и выхода не выполняются.

В качестве примера состояния рассмотрим ситуацию ввода пароля пользователя при аутентификации входа в некоторую программную систему (рис. 6.3). В этом случае список внутренних действий в данном состоянии не пуст и включает 4 отдельных действия, первые два из которых стандартные и описаны выше, а два последних определяются своей спецификацией.



**Рис. 6.3.** Пример состояния с непустой секцией внутренних действий

Начальное состояние

Начальное состояние представляет собой частный случай состояния, которое не содержит никаких внутренних действий (псевдосостояния). В этом состоянии находится объект по умолчанию в начальный момент времени. Оно служит для указания на диаграмме состояний графической области, от которой начинается процесс изменения состояний. Графически начальное состояние в языке UML обозначается в виде закрашенного кружка (рис. 6.4, а), из которого может только выходить стрелка, соответствующая переходу.



**Рис. 6.4.** Графическое изображение начального и конечного состояний на диаграмме состояний

На самом верхнем уровне представления объекта переход из начального состояния может быть помечен событием создания (инициализации) данного объекта. В противном случае переход никак не помечается. Если этот переход не помечен, то он является первым переходом в следующее за ним состояние.

Конечное состояние

Конечное (финальное) состояние представляет собой частный случай состояния, которое также не содержит никаких внутренних действий (псевдосостояния). В этом состоянии будет находиться объект по умолчанию после завершения работы автомата в конечный момент времени. Оно служит для указания на диаграмме состояний графической области, в которой завершается процесс изменения состояний или жизненный цикл данного объекта. Графически конечное состояние в языке UML обозначается в виде закрашенного кружка, помещенного в окружность (рис. 6.4, б), в которую может только входить стрелка, соответствующая переходу.

6.3. Переход

Простой переход (simple transition) представляет собой отношение между двумя последовательными состояниями, которое указывает на факт смены одного состояния другим. Пребывание моделируемого объекта в первом состоянии может сопровождаться выполнением некоторых действий, а переход во второе состояние будет возможен после завершения этих действий, а также после удовлетворения некоторых дополнительных условий. В этом случае говорят, что переход срабатывает, Или происходит срабатывание перехода. До срабатывания перехода объект находится в предыдущем от него состоянии, называемым исходным состоянием, или в источнике (не путать с начальным состоянием - это разные понятия), а после его срабатывания объект находится в последующем от него состоянии (целевом состоянии).

Переход осуществляется при наступлении некоторого события: окончания выполнения деятельности (do activity), получении объектом сообщения или приемом сигнала. На переходе указывается имя события Кроме того, на переходе могут указываться действия, производимые объектом в ответ на внешние события при переходе из одного состояния в другое. Срабатывание перехода может зависеть не только от наступления некоторого события, но и от выполнения определенного условия, называемого сторожевым условием. Объект перейдет из одного состояния в другое в том случае, если произошло указанное событие и сторожевое условие приняло значение "истина".

**Примечание**

Переход может быть направлен в то же состояние, из которого он выходит. В этом случае его называют переходом в себя. Исходное и целевое состояния перехода в себя совпадают. Этот переход изображается петлей со стрелкой и отличается от внутреннего перехода. При переходе в себя объект покидает исходное состояние, а затем снова входит в него. При этом всякий раз выполняются внутренние действия, специфицированные метками entry и exit.

На диаграмме состояний переход изображается сплошной линией со стрелкой, которая направлена в целевое состояние (например, "выход из строя" на рис. 6.1). Каждый переход может помечен строкой текста, которая имеет следующий общий формат:

<сигнатура события>'['<сторожевое условие>']' <выражение действия>.

При этом сигнатура события описывает некоторое событие с необходимыми аргументами:

<имя события>'('<список параметров, разделенных запятыми>')'.

Событие

Термин событие (event) требует отдельного пояснения, поскольку является самостоятельным элементом языка UML. Формально, событие представляет собой спецификацию некоторого факта, имеющего место в пространстве и во времени. Про события говорят, что они "происходят", при этом отдельные события должны быть упорядочены во времени. После наступления некоторого события нельзя уже вернуться к предыдущим событиям, если такая возможность не предусмотрена явно в модели.

Семантика понятия события фиксирует внимание на внешних проявлениях качественных изменений, происходящих при переходе моделируемого объекта из состояния в состояние. Например, при включении электрического переключателя происходит некоторое событие, в результате которого комната становится освещенной. После успешного ремонта компьютера также происходит немаловажное событие - восстановление его работоспособности. Если поднять трубку обычного телефона, то, в случае его исправности, мы ожидаем услышать тоновый сигнал. И этот факт тоже является событием.

В языке UML события играют роль стимулов, которые инициируют переходы из одних состояний в другие. В качестве событий можно рассматривать сигналы, вызовы, окончание фиксированных промежутков времени или моменты окончания выполнения определенных действий. Имя события идентифицирует каждый отдельный переход на диаграмме состояний и может содержать строку текста, начинающуюся со строчной буквы. В этом случае принято считать переход триггерным, т. е. таким, который специфицирует событие-триггер. Например, переходы на рис. 6.1 являются триггерными, поскольку с каждым из них связано некоторое событие-триггер, происходящее асинхронно в момент выхода из строя технического устройства или в момент окончания его ремонта.

Если рядом со стрелкой перехода не указана никакая строка текста, то соответствующий переход является нетриггерным, и в этом случае из контекста диаграммы состояний должно быть ясно, после окончания какой деятельности он срабатывает. После имени события могут следовать круглые скобки для явного задания параметров соответствующего события-триггера. Если таких параметров нет, то список параметров со скобками может отсутствовать.

***Упражнение:***

### Создание диаграммы

1. Найдите в броузере класс Order.
2. Щелкните на классе правой кнопкой мыши и в открывшемся меню укажите пункт Open State Diagram.

### Добавление начального и конечного состояний

1. На панели инструментов нажмите кнопку Start State (Начальное состояние).
2. Поместите это состояние на диаграмму.
3. На панели инструментов нажмите кнопку End State (Конечное состояние).
4. Поместите это состояние на диаграмму.

### Добавление суперсостояния

1. На панели инструментов нажмите кнопку State (Состояние).
2. Поместите это состояние на диаграмму.

### Добавление оставшихся состояний

1. На панели инструментов нажмите кнопку State (Состояние).
2. Поместите это состояние на диаграмму.
3. Назовите состояние Cancelled (Отменен).
4. На панели инструментов нажмите кнопку State (Состояние).
5. Поместите это состояние на диаграмму.
6. Назовите состояние Filled (Выполнен).
7. На панели инструментов нажмите кнопку State (Состояние).
8. Поместите это состояние на диаграмму внутрь суперсостояния.
9. Назовите состояние Initialization (Инициализация).
10. На панели инструментов нажмите кнопку State (Состояние).
11. Поместите это состояние на диаграмму внутрь суперсостояния.
12. Назовите состояние Pending (Выполнение заказа приостановлено).

### Подробное описание состояний

1. Дважды щелкните на состоянии Initialization (Инициализация).
2. Щелкните правой кнопкой мыши на окне Actions (Действия).
3. В открывшемся меню выберите пункт Insert (Вставить).
4. Дважды щелкните мышью на новом действии.
5. Назовите его Store Order Date (Сохранить дату заказа).
6. Убедитесь, что в окне When (Когда) указан пункт On Entry (На входе).
7. Повторите этапы 3 - 7, добавив следующие действия:

# Collect Customer Info (Собрать клиентскую информацию), в окне When указать пункт Do

# Add Order Items (Добавить к заказу новые графы), в окне When указать Do

1. Нажмите на кнопки ОК два раза, чтобы закрыть спецификацию.
2. Дважды щелкните на состоянии Cancelled (Отменен).
3. Повторите этапы 2 - 7, добавив действие Store Cancellation Data (Сохранить дату отмены), указать пункт On Exit (на выходе)
4. Нажмите на кнопки ОК два раза, чтобы закрыть спецификацию.
5. Дважды щелкните на состоянии Filled (Выполнен).
6. Повторите этапы 2 - 7, добавив действие Bill Customer (Выписать счет), указать пункт Do.
7. Нажмите на кнопки ОК два раза, чтобы закрыть спецификацию.

### Добавление переходов

1. На панели инструментов нажмите кнопку Transition (Переход).
2. Щелкните мышью на начальном состоянии.
3. Проведите линию перехода к состоянию Initialization (Инициализация).
4. Повторите этапы с первого по третий, создав следующие переходы:

# От состояния Initialization (Инициализация) к состоянию Pending (Выполнение заказа приостановлено)

# От состояния Pending (Выполнение заказа приостановлено) к состоянию Filled (Выполнен)

# От суперсостояния к состоянию Cancelled (Отменен)

# От состояния Cancelled (Отменен) к конечному состоянию

# От состояния Filled (Выполнен) к конечному состоянию

1. На панели инструментов нажмите кнопку Transition to Self (Переход к себе).
2. Щелкните на состоянии Pending (Выполнение заказа приостановлено).

### Подробное описание переходов

1. Дважды щелкните на переходе от состояния Initialization (Инициализация) к состоянию Pending (Выполнение заказа приостановлено), открыв окно его спецификации.
2. В поле Event (Событие) введите фразу Finalize order (Выполнить заказ).
3. Щелкните на кнопке ОК, закрыв окно спецификации.
4. Повторите этапы с первого по третий, добавив событие Cancel Order (Отменить заказ) к переходу между суперсостоянием и состоянием Cancelled (Отменен).
5. Дважды щелкните на переходе от состояния Pending (Выполнение заказа приостановлено) к состоянию Filled (Выполнен), открыв окно его спецификации.
6. В поле Event (Событие) введите фразу Add Order Item (Добавить к заказу новую позицию).
7. Перейдите на вкладку Detail (Подробно).
8. В поле Condition (Условие) введите No unfilled items remaining (Не осталось незаполненных позиций).
9. Щелкните на кнопке ОК, закрыв окно спецификации.
10. Дважды щелкните мышью на рефлексивном переходе (Transition to Self) состояния Pending (Выполнение заказа приостановлено).
11. В поле Event (Событие) введите фразу Add Order Item (Добавить к заказу новую позицию).
12. Перейдите на вкладку Detail (Подробно).
13. В поле Condition (Условие) введите Unfilled items remaining (Остаются незаполненные позиции).
14. Щелкните на кнопке ОК, закрыв окно спецификации.

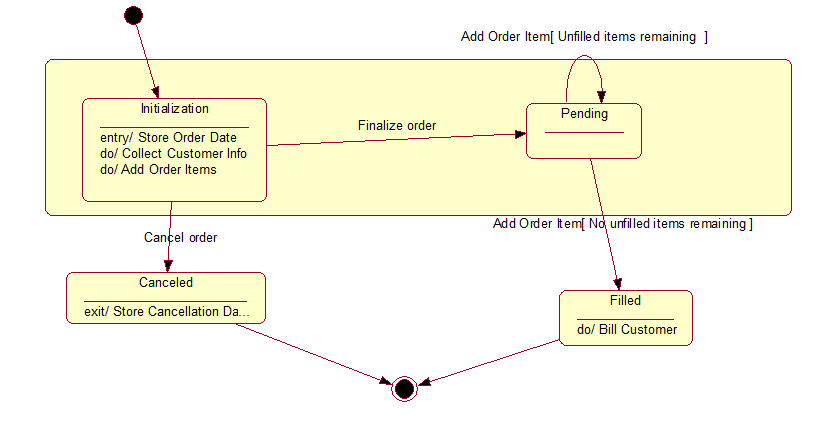


Рис 9. Диаграмма состояний

# **Лабораторная работа №7**. **Создание диаграммы компонентов**

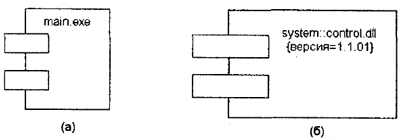
***Цель работы:***

* изучение диаграмм состояний,
* изучение их применения в процессе проектирования.

***Теория:***

Для представления физических сущностей в языке UML применяется специальный термин - компонент (component). Компонент реализует некоторый набор интерфейсов и служит для общего обозначения элементов физического представления модели. Для графического представления компонента может использоваться специальный символ - прямоугольник со вставленными слева двумя более мелкими прямоугольниками (рис. 10.1). Внутри объемлющего прямоугольника записывается имя компонента и, возможно, некоторая дополнительная информация. Изображение этого символа может незначительно варьироваться в зависимости от характера ассоциируемой с компонентом информации.

В метамодели языка UML компонент является потомком классификатора. Он предоставляет организацию в рамках физического пакета ассоциированным с ним элементам модели. Как классификатор, компонент может иметь также свои собственные свойства, такие как атрибуты и операции.



**Рис. 10.1.** Графическое изображение компонента в языке UML

Так, в первом случае (рис. 10.1, а) с компонентом уровня экземпляра связывается только его имя, а во втором (рис. 10.1, б) - дополнительно имя пакета и помеченное значение.

**Примечание**

Изображение компонента ведет свое происхождение от обозначения модуля программы, применявшегося некоторое время для отображения особенностей инкапсуляции данных и процедур. Так, верхний маленький прямоугольник концептуально ассоциируется с данными, которые реализует этот компонент (ранее он изображался в форме овала). Нижний маленький прямоугольник ассоциируется с операциями или методами, реализуемыми компонентом. В простых случаях имена данных и методов записывались явно в этих маленьких прямоугольниках, однако в языке UML они не указываются.

  Имя компонента

Имя компонента подчиняется общим правилам именования элементов модели в языке UML и может состоять из любого числа букв, цифр и некоторых знаков препинания. Отдельный компонент может быть представлен на уровне типа или на уровне экземпляра. Хотя его графическое изображение в обоих случаях одинаковое, правила записи имени компонента несколько отличаются. Если компонент представляется на уровне типа, то в качестве его имени записывается только имя типа с заглавной буквы.

Если же компонент представляется на уровне экземпляра, то в качестве его имени записывается <имя компонента ':' имя типаХ При этом вся строка имени подчеркивается.

**Примечание**

Хотя правила именования объектов в языке UML требуют подчеркивания имени отдельных экземпляров, применительно к компонентам в литературе подчеркивание их имени часто опускают. В этом случае запись имени компонента со строчной буквы будет характеризовать компонент уровня экземпляра.

В качестве простых имен принято использовать имена исполняемых файлов (с указанием расширения ехе после точки-разделителя), имена динамических библиотек (расширение dll), имена Web-страниц (расширение html), имена текстовых файлов (расширения txt или doc) или файлов справки (hip), имена файлов баз данных (DB) или имена файлов с исходными текстами программ (расширения h, cpp для языка C++, расширение Java для языка Java), скрипты (pi, asp) и др.

Поскольку конкретная реализация логического представления модели системы зависит от используемого программного инструментария, то и имена компонентов будут определяться особенностями синтаксиса соответствующего языка программирования.

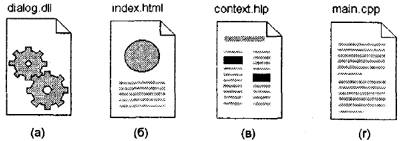
В отдельных случаях к простому имени компонента может быть добавлена информация об имени объемлющего пакета и о конкретной версии реализации данного компонента (рис. 10.1, б). Необходимо заметить, что в этом случае номер версии записывается как помеченное значение в фигурных скобках. В других случаях символ компонента может быть разделен на секции, чтобы явно указать имена реализованных в нем интерфейсов. Такое обозначение компонента называется расширенным и рассматривается ниже в этой главе.

  Виды компонентов

Поскольку компонент как элемент физической реализации модели представляет отдельный модуль кода, иногда его комментируют с указанием дополнительных графических символов, иллюстрирующих конкретные особенности его реализации. Строго.говоря, эти дополнительные обозначения для примечаний не специфицированы в языке UML. Однако их применение упрощает понимание диаграммы компонентов, существенно повышая наглядность физического представления. Некоторые из таких общепринятых обозначений для компонентов изображены ниже (рис. 10.2).

В языке UML выделяют три вида компонентов.

* Во-первых, компоненты развертывания, которые обеспечивают непосредственное выполнение системой своих функций. Такими компонентами могут быть динамически подключаемые библиотеки с расширением dll (рис. 10.2, а), Web-страницы на языке разметки гипертекста с расширением html (рис. 10.2, б) и файлы справки с расширением Ыр (рис. 10.2, в).
* Во-вторых, компоненты-рабочие продукты. Как правило - это файлы с исходными текстами программ, например, с расширениями h или срр для языка C++ (рис. 10.2, г).
* В-третьих, компоненты исполнения, представляющие исполнимые модули - файлы с расширением ехе. Они обозначаются обычным образом.



**Рис. 10.2.** Варианты графического изображения компонентов на диаграмме компонентов

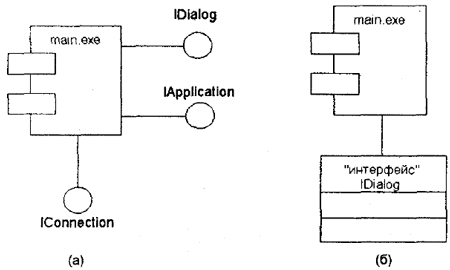
Эти элементы иногда называют артефактами, подчеркивая при этом их законченное информационное содержание, зависящее от конкретной технологии реализации соответствующих компонентов. Более того, разработчики могут для этой цели использовать самостоятельные обозначения, поскольку в языке UML нет строгой нотации для графического представления примечаний.

Другой способ спецификации различных видов компонентов - явное указание стереотипа компонента перед его именем. В языке UML для компонентов определены следующие стереотипы:

* Библиотека (library) - определяет первую разновидность компонента, который представляется в форме динамической или статической библиотеки.
* Таблица (table) - также определяет первую разновидность компонента, который представляется в форме таблицы базы данных.
* Файл (file) - определяет вторую разновидность компонента, который представляется в виде файлов с исходными текстами программ.
* Документ (document) - определяет вторую разновидность компонента, . который представляется в форме документа.
* Исполнимый (executable) - определяет третий вид компонента, который может исполняться в узле.

  10.2. Интерфейсы

Следующим элементом диаграммы компонентов являются интерфейсы. Последние уже неоднократно рассматривались ранее, поэтому здесь будут отмечены те их, особенности, которые характерны для представления на диаграммах компонентов. Напомним, что в общем случае интерфейс графически изображается окружностью, которая соединяется с компонентом отрезком линии без стрелок (рис. 10.3, а). При этом имя интерфейса, которое обязательно должно начинаться с заглавной буквы "I", записывается рядом с окружностью. Семантически линия означает реализацию интерфейса, а наличие интерфейсов у компонента означает, что данный компонент реализует соответствующий набор интерфейсов.



**Рис. 10.3.** Графическое изображение интерфейсов на диаграмме компонентов

Другим способом представления интерфейса на диаграмме компонентов является его изображение в виде прямоугольника класса со стереотипом "интерфейс" и возможными секциями атрибутов и операций (рис. 10.3, б). Как правило, этот вариант обозначения используется для представления внутренней структуры интерфейса, которая может быть важна для реализации.

При разработке программных систем интерфейсы обеспечивают не только совместимость различных версий, но и возможность вносить существенные изменения в одни части программы, не изменяя другие ее части. Таким образом, назначение интерфейсов существенно шире, чем спецификация взаимодействия с пользователями системы (актерами).

**Примечание**

Характер использования интерфейсов отдельными компонентами может отличаться. Поэтому различают два способа связи интерфейса и компонента. Если компонент реализует некоторый интерфейс, то такой интерфейс называют экспортируемым, поскольку этот компонент предоставляет его в качестве сервиса другим компонентам. Если же компонент использует некоторый интерфейс, который реализуется другим компонентом, то такой интерфейс для первого компонента называется импортируемым. Особенность импортируемого интерфейса состоит в том, что на диаграмме компонентов это отношение изображается с помощью зависимости.

## *Упражнение:*

### Создание пакетов компонентов

1. Щелкните правой кнопкой мыши на представлении компонентов в броузере.
2. В открывшемся меню выберите пункт New > Package (Создать > пакет).
3. Назовите этот пакет Entities (Сущности).
4. Повторите этапы с первого по третий, создав пакеты Boundaries (Границы) и Control (Управление).

### Добавление пакетов на Главную диаграмму Компонентов

1. Откройте Главную диаграмму Компонентов, дважды щелкнув на ней.
2. Перетащите пакеты Entities, Boundary и Control из броузера на Главную диаграмму.

### Рисование зависимостей между пакетами

1. На панели инструментов нажмите кнопку Dependency (Зависимость).
2. Щелкните мышью на упаковке Boundaries Главной диаграммы Компонентов.
3. Проведите линию зависимости до упаковки Control.
4. Повторите этапы 1 - 3, проведя еще зависимость от пакета Control до пакета Entities.

### Добавление компонентов к пакетам и рисование зависимостей

1. Дважды щелкните мышью на пакете Entities Главной диаграммы Компонентов, открыв Главную диаграмму Компонентов этого пакета.
2. На панели инструментов нажмите кнопку Package Specification (Спецификация пакета).
3. Поместите спецификацию пакета на диаграмму.
4. Введите имя спецификации пакета OrderItem.
5. Повторите этапы 2 - 4, добавив спецификацию пакета Order.
6. На панели инструментов нажмите кнопку Package Body (Тело пакета).
7. Поместите его на диаграмму.
8. Введите имя тела пакета OrderItem.
9. Повторите этапы 6 - 8, добавив тело пакета Order.
10. На панели инструментов нажмите кнопку Dependency (Зависимость).
11. Щелкните мышью на теле пакета OrderItem.
12. Проведите линию зависимости от него к спецификации пакета OrderItem.
13. Повторите этапы 10 - 12, добавив линию зависимости между телом пакета Order и спецификацией пакета Order.
14. Повторите этапы 10 - 12, добавив линию зависимости от спецификации пакета Order к спецификации пакета OrderItem.
15. С помощью описанного метода создайте следующие компоненты и зависимости:

Для пакета Boundaries:

# Спецификацию пакета OrderOptions

# Тело пакета OrderOptions

# Спецификацию пакета OrderDetail

# Тело пакета OrderDetail

Зависимости в пакете Boundaries:

# От тела пакета OrderOptions до спецификации пакета OrderOptions

# От тела пакета OrderDetail до спецификации пакета OrderDetail

# От спецификации пакета OrderOptions до спецификации пакета OrderDetail

Для пакета Control:

# Спецификацию пакета OrderMgr

# Тело пакета OrderMgr

# Спецификацию пакета TransactionMgr

# Тело пакета TransactionMgr

Зависимости в пакете Control:

# От тела пакета OrderMgr до спецификации пакета OrderMgr

# От тела пакета TransactionMgr до спецификации пакета TransactionMgr

# От спецификации пакета OrderMgr до спецификации пакета TransactionMgr

### Создание диаграммы Компонентов системы

1. Щелкните правой кнопкой мыши на представлении Компонентов в броузере.
2. В открывшемся меню выберите пункт New > Component Diagram
3. Назовите новую диаграмму System.
4. Дважды щелкните на этой диаграмме.

### Размещение компонентов на диаграмме Компонентов системы

1. Если это еще не было сделано, разверните в броузере пакет компонентов Entities, чтобы открыть его.
2. Щелкните мышью на спецификации пакета Order в пакете компонентов Entities.
3. Перетащите эту спецификацию на диаграмму.
4. Повторите этапы 2 и 3, поместив на диаграмму спецификацию пакета OrderItem.
5. С помощью этого метода поместите на диаграмму следующие компоненты:

Из пакета компонентов Boundaries:

# Спецификацию пакета OrderOptions

# Спецификацию пакета OrderDetail

Из пакета компонентов Control:

# Спецификацию пакета OrderMgr

# Спецификацию пакета TransactionMgr

1. На панели инструментов нажмите кнопку Task Specification (Спецификация задачи).
2. Поместите спецификацию задачи на диаграмму и назовите ее OrderClientExe.
3. Повторите этапы 6 и 7 для спецификации задачи OrderServerExe.

### Добавление оставшихся зависимостей на диаграмму Компонентов системы

Уже существующие зависимости будут автоматически показаны на диаграмме Компонентов системы после добавления туда соответствующих компонентов. Теперь надо добавить остальные зависимости.

1. На панели инструментов нажмите кнопку Dependency (Зависимость).
2. Щелкните на спецификации пакета OrderDetail.
3. Проведите линию зависимости к спецификации пакета OrderMgr.
4. Повторите этапы 1 - 3, создав следующие зависимости:

# От спецификации пакета OrderMgr к спецификации пакета Order

# От спецификации пакета TransactionMgr к спецификации пакета OrderItem

# От спецификации пакета TransactionMgr к спецификации пакета Order

# От спецификации задачи OrderClientExe к спецификации пакета OrderOptions

# От спецификации задачи OrderServerExe к спецификации пакета OrderMgr

### Соотнесение классов с компонентами

1. В Логическом представлении броузера найдите класс Order пакета Entities.
2. Перетащите этот класс на спецификацию пакета компонента Order в представлении Компонентов броузера. В результате класс Order будет соотнесен со спецификацией пакета компонента Order.
3. Перетащите класс Order на тело пакета компонента Order в представлении Компонентов броузера. В результате класс Order будет соотнесен с телом пакета компонента Order.
4. Повторите этапы 1 - 3, соотнеся с классами следующие компоненты:

# Класс OrderItem со спецификацией пакета OrderItem

# Класс OrderItem с телом пакета OrderItem

# Класс OrderOptions со спецификацией пакета OrderOptions

# Класс OrderOptions с телом пакета OrderOptions

# Класс OrderDetail со спецификацией пакета OrderDetail

# Класс OrderDetail с телом пакета OrderDetail

# Класс OrderMgr со спецификацией пакета OrderMgr

# Класс OrderMgr с телом пакета OrderMgr

# Класс TransactionMgr со спецификацией пакета TransactionMgr

# Класс TransactionMgr с телом пакета TransactionMgr

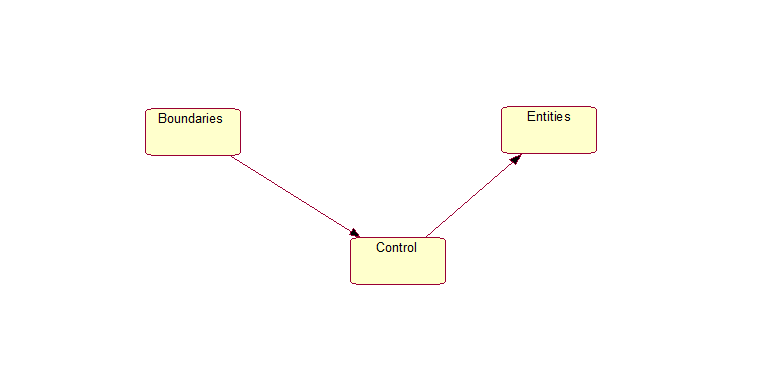


Рис 10. Главная диаграмма компонентов системы

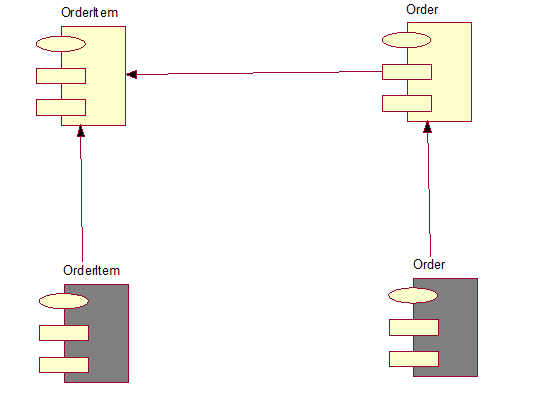


Рис 11. Диаграмма Компонентов пакета Entities.

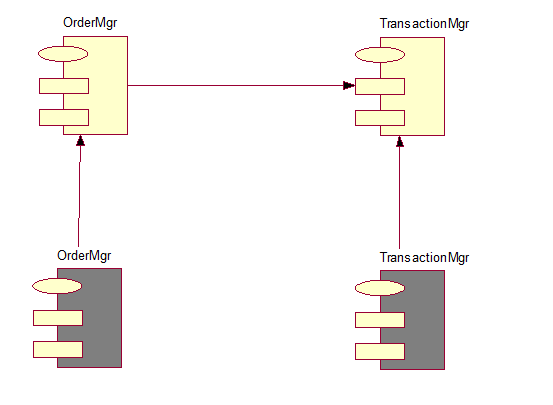


Рис 12. Диаграмма Компонентов пакета Сontrol

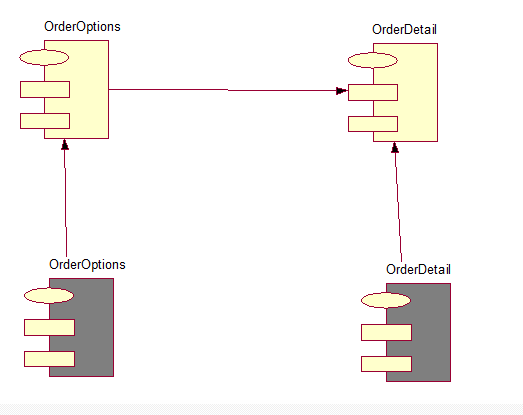
.

Рис 13. Диаграмма Компонентов пакета Boundaries

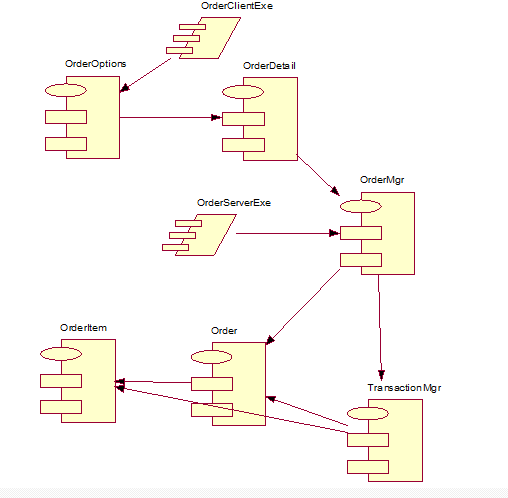


Рис 14. Диаграмма Компонентов системы

# **Лабораторная работа №8. Создание диаграммы размещения**

***Цель работы:***

* изучение диаграмм размещения,
* изучение их применения в процессе проектирования.

***Теория:***

Физическое представление программной системы не может быть полным, если отсутствует информация о том, на какой платформе и на каких вычислительных средствах она реализована. Конечно, если разрабатывается простая программа, которая может выполняться локально на компьютере пользователя, не задействуя никаких периферийных устройств и ресурсов, то в этом случае нет необходимости в разработке дополнительных диаграмм. Однако при разработке корпоративных приложений ситуация представляется совсем по-другому.

Во-первых, сложные программные системы могут реализовываться в сетевом варианте на различных вычислительных платформах и технологиях доступа к распределенным базам данных. Наличие локальной корпоративной сети требует решения целого комплекса дополнительных задач по рациональному размещению компонентов по узлам этой сети, что определяет общую производительность программной системы.

Во-вторых, интеграция программной системы с Интернетом определяет необходимость решения дополнительных вопросов при проектировании системы, таких как обеспечение безопасности, криптозащищенности и устойчивости доступа к информации для корпоративных клиентов. Эти аспекты в немалой степени зависят от реализации проекта в форме физически существующих узлов системы, таких как серверы, рабочие станции, брандмауэры, каналы связи и хранилища данных.

Наконец, технологии доступа и манипулирования данными в рамках общей схемы "клиент-сервер" также требуют размещения больших баз данных в различных сегментах корпоративной сети, их резервного копирования, архивирования, кэширования для обеспечения необходимой производительности системы в целом. Эти аспекты также требуют визуального представления с целью спецификации программных и технологических особенностей реализации распределенных архитектур.

Как было отмечено в главе 10, первой из диаграмм физического представления является диаграмма компонентов. Второй формой физического представления программной системы является диаграмма развертывания (синоним - диаграмма размещения). Она применяется для представления общей конфигурации и топологии распределенной программной системы и содержит распределение компонентов по отдельным узлам системы. Кроме того, диаграмма развертывания показывает наличие физических соединений - маршрутов передачи информации между аппаратными устройствами, задействованными в реализации системы.

Диаграмма развертывания предназначена для визуализации элементов и компонентов программы, существующих лишь на этапе ее исполнения (runtime). При этом представляются только компоненты-экземпляры программы, являющиеся исполнимыми файлами или динамическими библиотеками. Те компоненты, которые не используются на этапе исполнения, на диаграмме развертывания не показываются. Так, компоненты с исходными текстами программ могут присутствовать только на диаграмме компонентов. На диаграмме развертывания они не указываются.

Диаграмма развертывания содержит графические изображения процессоров, устройств, процессов и связей между ними. В отличие от диаграмм логического представления, диаграмма развертывания является единой для системы в целом, поскольку должна всецело отражать особенности ее реализации. Эта диаграмма, по сути, завершает процесс ООАП для конкретной программной системы и ее разработка, как правило, является последним этапом спецификации модели.

Итак, перечислим цели, преследуемые при разработке диаграммы развертывания:

* Определить распределение компонентов системы по ее физическим узлам.
* Показать физические связи между всеми узлами реализации системы на этапе ее исполнения.
* Выявить узкие места системы и реконфигурировать ее топологию для достижения требуемой производительности.

Для обеспечения этих требований диаграмма развертывания разрабатывается совместно системными аналитиками, сетевыми инженерами и системотехниками. Далее рассмотрим отдельные элементы, из которых состоят диаграммы развертывания.

***Упражнение:***

### Добавление узлов к диаграмме Размещения

1. Дважды щелкните мышью на представлении Размещения в броузере, чтобы открыть диаграмму Размещения.
2. На панели инструментов нажмите кнопку Processor (Процессор).
3. Щелкните на диаграмме, поместив туда процессор.
4. Введите имя процессора "Сервер базы данных".
5. Повторите этапы 2 - 4, добавив следующие процессоры:

# Сервер приложения

# Клиентская рабочая станция №1

# Клиентская рабочая станция №2

1. На панели инструментов нажмите кнопку Device (Устройство).
2. Щелкните на диаграмме, поместив на нее устройство.
3. Назовите его "Принтер".

### Добавление связей

1. На панели инструментов нажмите кнопку Connection (Связь).
2. Щелкните на процессоре "Сервер базы данных".
3. Проведите линию связи к процессору "Сервер приложения".
4. Повторите этапы 1 - 3, добавив следующие связи:

# От процессора "Сервер приложения" к процессору "Клиентская рабочая станция №1"

# От процессора "Сервер приложения" к процессору "Клиентская рабочая станция №2"

# От процессора "Сервер приложения" к устройству "Принтер"

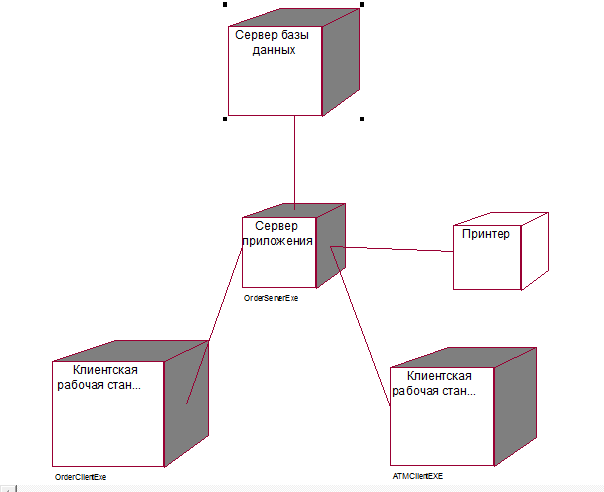


Рис 15. Диаграмма размещения

### Добавление процессов

1. Щелкните правой кнопкой мыши на процессоре "Сервер приложения" в броузере.
2. В открывшемся меню выберите пункт New > Process (Создать > Процесс).
3. Введите имя процесса OrderServerExe.
4. Повторите этапы 1 - 3, добавив еще процессы:

# На процессоре "Клиентская рабочая станция №1" -- процесс OrderClientExe

# На процессоре "Клиентская рабочая станция №2" -- процесс ATMClientEXE

### Показ процессов на диаграмме

1. Щелкните правой кнопкой мыши на процессоре "Сервер приложения".
2. В открывшемся меню выберите пункт Show Processes (Показать процессы).
3. Повторите этапы 1 и 2, показав процессы на следующих процессорах:

# Клиентская рабочая станция №1

# Клиентская рабочая станция №2.