КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Визуальное моделирование систем в StarUML

Казань 2013

УДК 004.4'22
519.682.6

Печатается по решению редакционно-издательского совета института вычислительной математики и информационных технологий ВМК Казанского (Приволжского) федерального университета

Рецензенты:
доктор технических наук, профессор КНИТУ-КАИ Гайнутдинов В.Г.
кандидат педагогических наук, доцент К(П)ФУ Халитова З.Р.

Каюмова А.В.
Визуальное моделирование систем в StarUML: Учебное пособие/ А.В. Каюмова. Казань. – Казанский федеральный университет, 2013. – 104с.

Предлагаемое пособие предназначено студентам специальности «прикладная информатика» для аудиторных и самостоятельных занятий по предметам «Проектирование информационных систем» и «Проектный практикум». В пособии описываются основные элементы нотации диаграмм UML, на конкретном примере рассматривается процесс визуального моделирования информационных систем с применением CASE-инструмента StarUML.

ОГЛАВЛЕНИЕ

Введение

Информационные технологии развиваются стремительно. В условиях жесткой конкуренции на рынке и растущих запросов пользователей создание информационных систем стало очень сложной задачей. Системы стали настолько велики, что физических и умственных возможностей и способностей человека уже просто не достаточно для того, чтобы спроектировать сложную систему за один шаг, и даже для того, чтобы просто представить, вообразить все возможности и потребности проектируемой системы, ее архитектуру и программное обеспечение. Однако замечено, что визуальная информация воспринимается наиболее успешно и полно. На помощь проектировщику сложных систем пришло визуальное моделирование.

Предлагаемое учебное пособие посвящено рассмотрению основных приемов визуального моделирования систем с помощью UML и предназначено студентам специальности «прикладная информатика» и др. для аудиторных и самостоятельных занятий по предметам «Проектирование информационных систем» и «Проектный практикум». В пособии описываются основные элементы нотации диаграмм UML, на конкретном примере рассматривается процесс проектирования информационных систем с применением программной платформы StarUML, приводятся некоторые приемы и способы создания моделей системы: поиск классов, их атрибутов и операций, поиск объектов системы и др. Этапы создания визуальной модели сопровождаются иллюстрированными инструкциями.

1. Визуальное моделирование и UML

Визуальным моделированием (visual modeling) называется способ представления идей и проблем реального мира с помощью моделей[1].

Модель – это абстракция, описывающая суть сложной проблемы или структуры без акцента на несущественных деталях, тем самым делая ее более понятной.

Разработка программного обеспечения - не исключение. При построении сложной системы строятся ее абстрактные визуальные модели.

В настоящее время в области проектирования информационных систем с успехом применяется визуальное моделирование с помощью унифицированного языка моделирования UML.

Унифицированный язык моделирования (Unified Modeling Language, UML) является графическим языком для визуализации, специфицирования, конструирования и документирования систем, в которых большая роль принадлежит программному обеспечению [2].

С помощью UML можно детально описать систему, начиная разработку с концептуальной модели с ее бизнес-функциями и процессами, а также описать особенности реализации системы, такие как классы программного обеспечения системы, схему базы данных. Используя UML, мы также можем разрабатывать сложные системы быстро и качественно.

Как язык графического визуального моделирования UML имеет свою **нотацию** – принятые обозначения. Нотация обеспечивает семантику языка, является способом унификации обозначений визуального моделирования, обеспечивает всестороннее представление системы, которое сравнительно легко и свободно воспринимается человеком. Последняя версия нотации UML 2.4.1 опубликована в августе 2011 года.

Моделирование с помощью UML осуществляется поэтапным построением ряда диаграмм, каждая из которых отражает какую-то часть или сторону системы либо ее замысла.

Диаграмма - это графическое представление множества элементов. Обычно диаграмма изображается в виде графа с вершинами (сущностями) и ребрами (отношениями). Диаграммы подчиняются нотации UML и изображаются в соответствии с ней.

Основные диаграммы UML:

- вариантов использования (*use case diagram*);
- классов (*class diagram*);

- кооперации (*collaboration diagram*);
- последовательности (*sequence diagram*);
- состояний (*statechart diagram*);
- деятельности (*activity diagram*);
- компонентов (*component diagram*);
- развертывания (*deployment diagram*).

Построения этих диаграмм достаточно для полного моделирования системы.

В данном пособии рассматриваются основные элементы нотации диаграмм и принципы их построения.

2. Выбор CASE-средства проектирования информационных систем.

UML - это язык визуального моделирования систем. Моделирование систем с помощью UML предполагает построение ряда взаимосвязанных диаграмм. Для сопровождения процесса построения, анализа и документирования модели, а также проверки модели и генерации программных кодов разработчики используют специально для этих целей созданные CASE-инструменты проектирования систем.

В общем смысле **CASE** (Computer-Aided Software Engineering) — это набор инструментов и методов программной инженерии для проектирования программного обеспечения, который помогает обеспечить высокое качество программ, отсутствие ошибок и простоту в обслуживании программных продуктов [8].

Существует достаточно много CASE-инструментов моделирования и проектирования систем и баз данных (не только с помощью UML). В данном учебном пособии для примера моделирования системы выбран программный инструмент моделирования StarUML [7].

Данная программная платформа имеет свободную лицензию и доступна для установки с официального сайта StarUML [7].

StarUML поддерживает одиннадцать различных типов диаграмм, принятых в нотации UML 2.0, а также подход MDA (модельно-настраиваемая архитектура), предлагает настройку параметров пользователя для адаптации среды разработки, поддерживает расширения, предоставляет различного рода модули, расширяющие возможности StarUML.

2.1 Создание нового проекта в StarUML

Основная структурная единица в StarUML – это проект. Проект сохраняется в одном файле в формате XML с расширением «.UML». Проект может содержать одну или несколько моделей и различные представления этих моделей (View) – визуальные выражения информации, содержащейся в моделях. Каждое представление модели содержит диаграммы – визуальные образы, отображающие определенные аспекты модели.

Новый проект будет автоматически создан при запуске программы StarUML. При этом вам будет предложено в диалоговом окне выбрать один из подходов (Approaches), поддерживаемых StarUML (см. рис. 1).

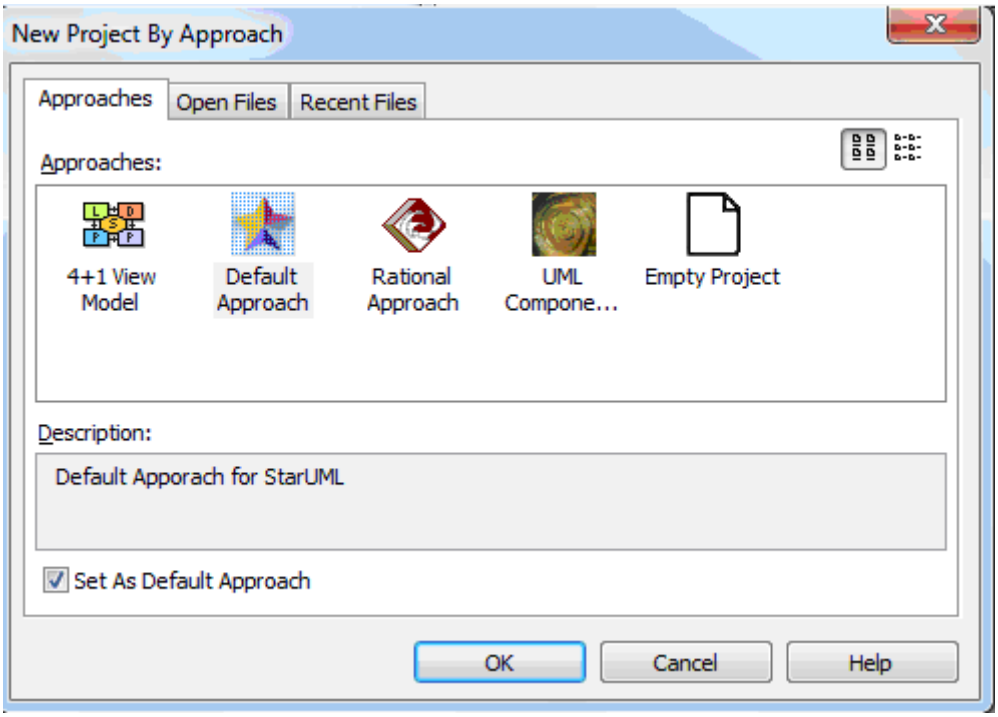


Рисунок 1. Выбор подхода

Существуют различные методологии моделирования информационных систем, компании-разработчики систем также могут разрабатывать свои методологии. Следовательно, на начальной стадии проектирования необходимо определить основные положения методологии или выбрать одну из уже существующих. Для того чтобы согласовать между собой различные элементы и этапы моделирования, StarUML предлагает концепцию подходов.

После того как мы выбрали один из предложенных подходов, появится основное окно программы (рис. 2).

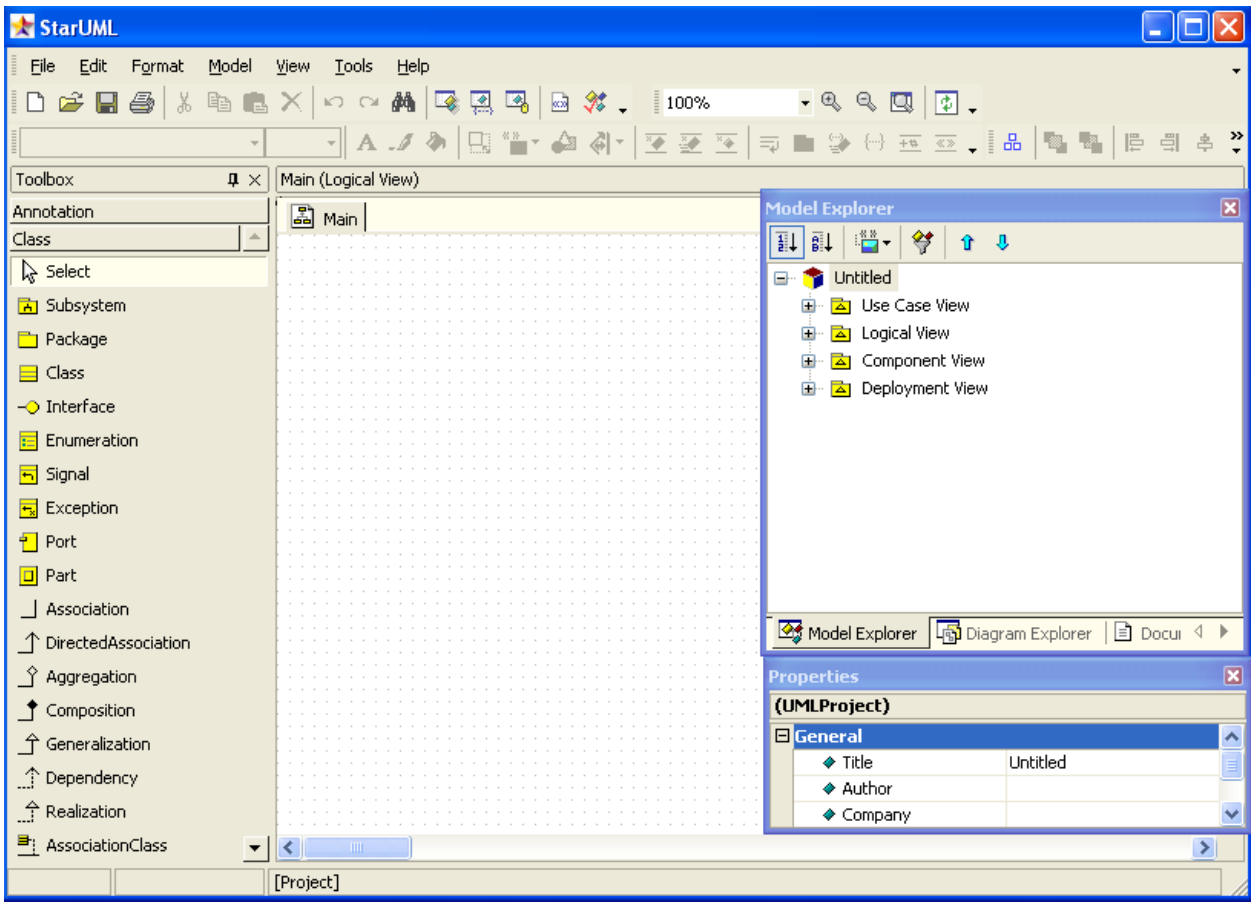


Рисунок 2. Основное окно программы

В верхней части окна расположено главное меню, кнопки быстрого доступа. Слева расположена панель элементов (Toolbox) с изображениями элементов диаграммы. Элементы соответствуют типу выбранной диаграммы. В центре находится рабочее поле диаграммы, на котором она может быть построена с использованием соответствующих элементов панели инструментов.

Справа находится **инспектор модели**, на котором можно найти вкладки навигатора модели Model Exploer, навигатора диаграмм Diagram Exploer, окно редактора свойств Properties, окно документирования элементов модели Documentation и редактор вложений Attachments. Внешний вид инспектора модели с вкладками представлен ниже (рис. 3).

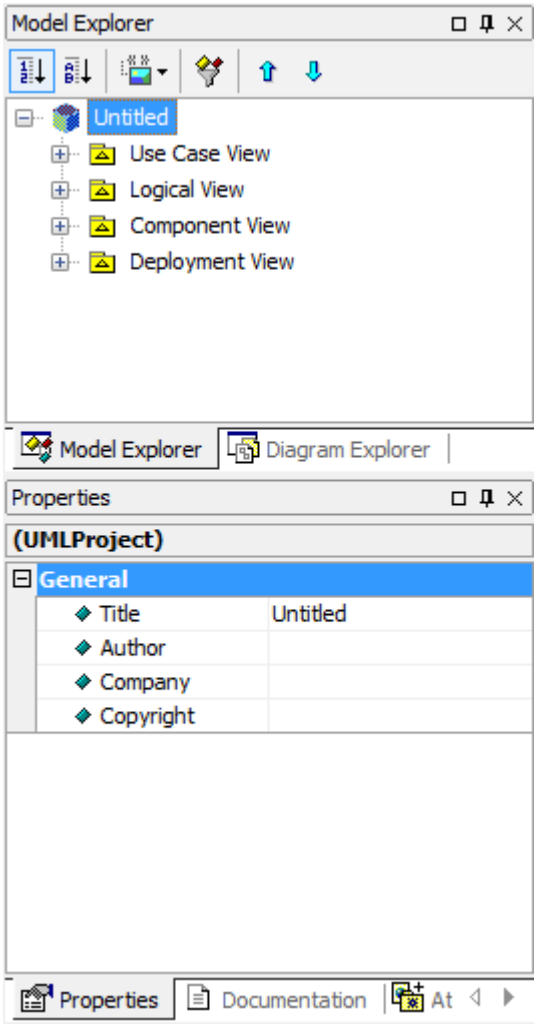


Рисунок 3. Инспектор модели

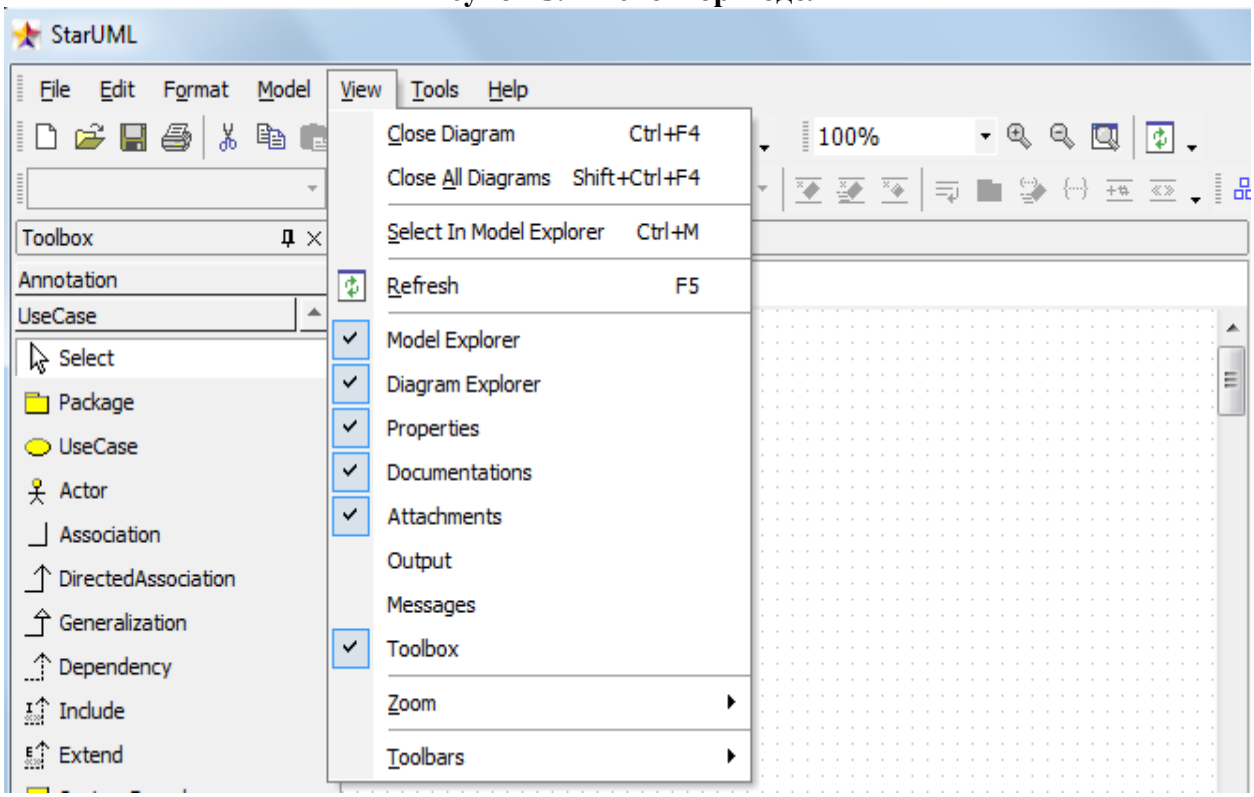


Рисунок 4. Пункт меню View. Управление видом инспектора модели

Управлять видом инспектора модели, панели элементов, закрывать и открывать редакторы инспектора можно с помощью пункта меню View (рис.4). Если рядом с пунктом меню стоит «галочка», этот элемент активен и его можно видеть в окне программы или открыть на доступных вкладках инспектора модели.

Иерархическая структура проекта отображается справа на навигаторе модели (Model Exploer). В зависимости от выбранного подхода на навигаторе модели будут отображены различные пакеты представлений модели. Каждый пакет представления будет содержать элементы моделей и диаграмм, которые мы создадим.

Если при создании нового проекта моделирования мы выберем подход Rational Approach, то при таком подходе в навигаторе будут присутствовать четыре пакета представлений модели системы (см. рис. 5):

- Use Case View – представление требований к системе, описывает, что система должна делать;
- Logical View – логическое представление системы, описывает, как система должна быть построена;
- Component View – представление реализации, описывает зависимость между программными компонентами;
- Deployment View – представление развертывания, описывает аппаратные элементы, устройства и программные компоненты.

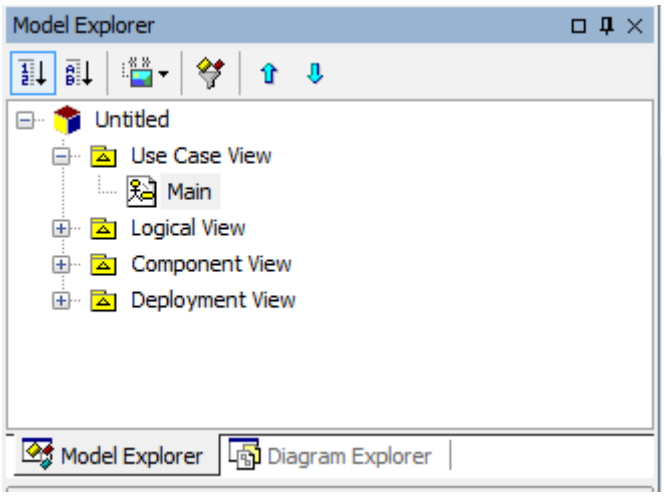


Рисунок 5. Навигатор модели

Сейчас каждое представление содержит одну диаграмму с именем Main. Если щелкнуть по ней два раза, то откроется рабочее поле этой диаграммы и соответствующая панель инструментов.

Пример. Если щелкнуть два раза по диаграмме Main представления Use Case View, то откроется рабочее этой диаграммы и ее панель элементов (рис. 6).

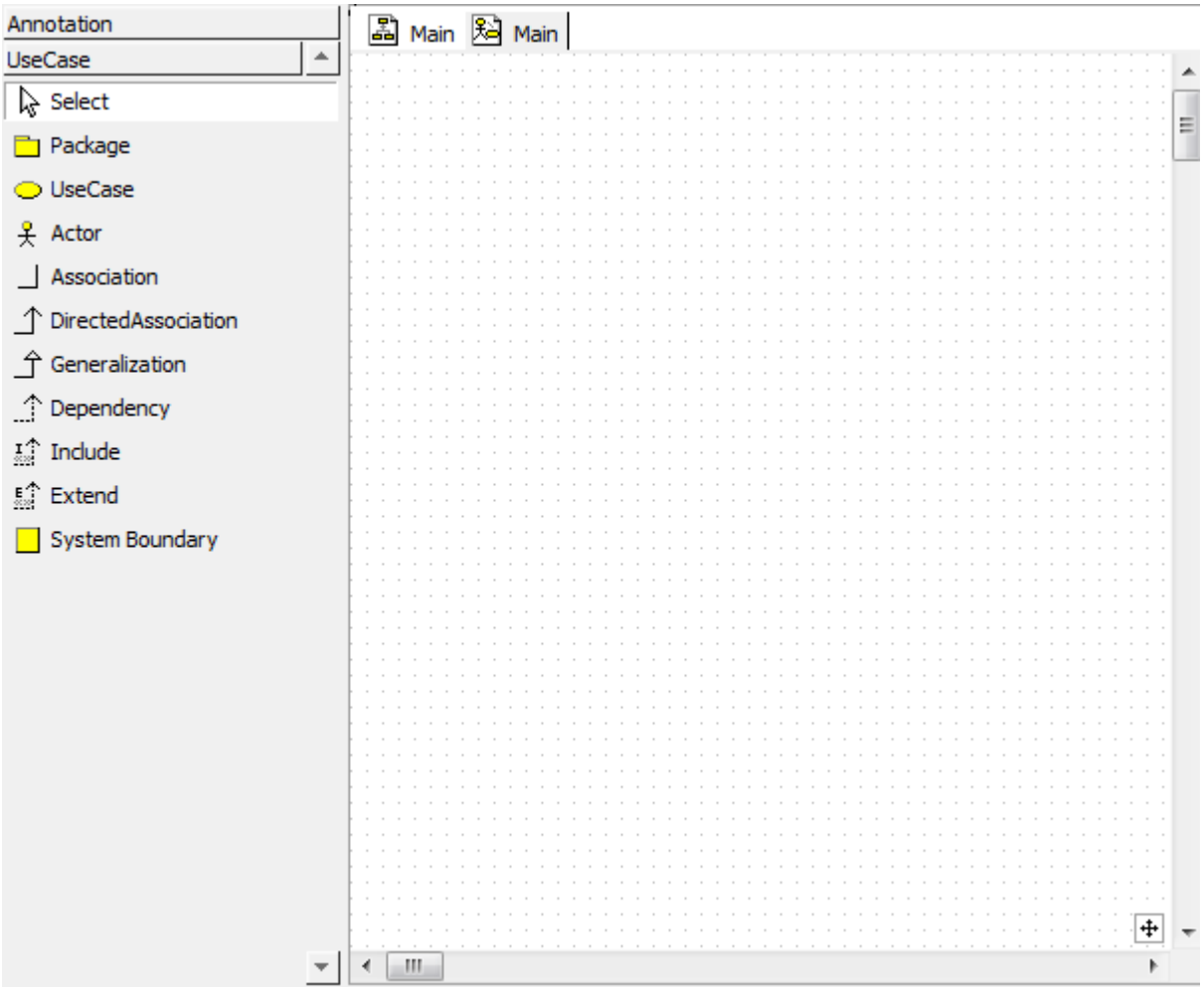


Рисунок 6. Рабочее поле диаграммы прецедентов Main и ее панель элементов

Ниже иерархии представлений отображаются свойства выделенного элемента модели или диаграммы (в данном случае свойства диаграммы Main, так как она выделена в навигаторе модели) (см. рис. 7).

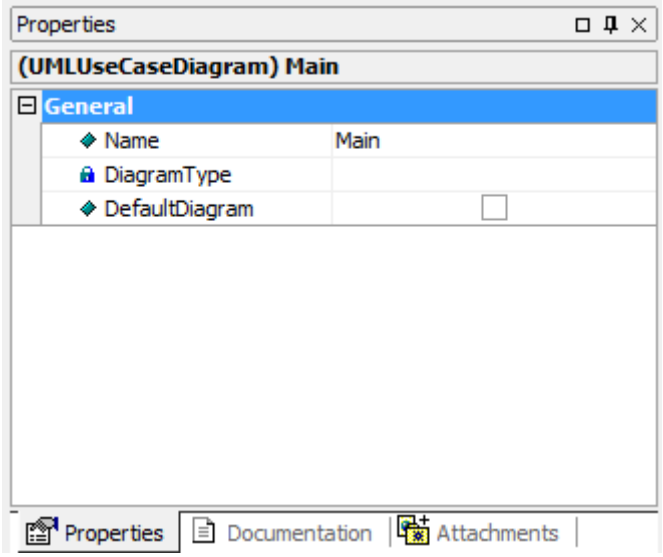


Рисунок 7. Редактор свойств

Рассмотрим визуальное моделирование систем с использованием UML на примере проектирования системы заказов интернет-магазина «Style».

3. Постановка задачи. Определение рабочей области моделирования

Магазин занимается продажей детской и взрослой одежды и обуви различных брендов. Покупатель просматривает каталог и делает заказ. Предполагаем, что потенциальный клиент заходит на сайт магазина, он может нажать кнопку просмотра (или загрузки) каталога, далее может положить понравившийся товар в корзину, изменить корзину и, приняв решение о покупке товаров, перейти из корзины к оформлению заказа.

Для того чтобы корректно создать систему, отвечающую всем требованиям заказчика, мы должны абсолютно четко представить себе ее основные бизнес-функции и выяснить предъявляемые к системе требования. Для этого необходимо провести обследование компании и построить ее полную бизнес-модель. Поскольку наш пример является придуманным, мы не можем провести такое обследование и не имеем возможности общаться с заказчиком, то мы будем опираться на придуманное нами словесное описание системы.

3.1 Описание работы системы

Каждый товар в каталоге описывается артикулом, размерным рядом, ценой и фото с кратким описанием.

Покупатель может загрузить каталог товаров. Каталог не содержит разделы, имеет блочную структуру, состоит из набора товаров с фото, ценой и размерами. Покупатель складывает понравившиеся товары в корзину, при этом выбирая размер и количество необходимого товара данного артикула.

Корзину можно изменить: просмотреть, удалить товар, изменить количество позиций одного артикула, вернуться в каталог.

Когда покупатель делает заказ, он вводит свои личные данные, телефон и оплачивает его по банковской карте (если заказ не оплачен, то он и не сделан).

После того как сделан заказ, его можно забрать со склада через 1 рабочий день. Данные о заказе поступают сотруднику магазина, назовем его сотрудником отдела продаж, он проверяет наличие товаров и передает его кладовщику на комплектацию. Кладовщик, собрав заказ, делает отметку о готовности.

Заказ выдается со склада кладовщиком. Кладовщик выдает заказ и отмечает в системе, что заказ выдан.

Магазин не занимается доставкой заказов, не делает скидок. Для того

чтобы ограничить масштаб задачи, мы не рассматриваем систему снабжения магазина новыми товарами. Этим занимается другая система, назовем ее Склад. Информация о проданных товарах (т.е. сделанных заказах) поступает также в систему Склад.

Поскольку на протяжении от создания до выдачи заказа, он проходит разные стадии, то будет разумно ввести понятие статуса заказа. Сотрудники магазина могут статус заказа изменять, а покупатель может проследить за сборкой заказа. В таком случае наша система предоставляет еще одну функцию: узнать статус заказа.

3.2 Создание проекта

Создадим новый проект в StarUML, выбрав Rational Approach из списка предложенных подходов. Наша модель будет иметь четыре представления: Use Case, Logical, Component и Deployment. Данный подход по структуре представлений на наш взгляд наиболее соответствует методологии Rational Unified Procces (RUP), которая поддерживает итеративный процесс разработки информационных систем. Подробные сведения об этой методологии можно найти в [2] и [4]. Настоящее пособие не претендует на полное соответствие этапам RUP: целью моделирования нашей системы является знакомство с нотацией UML и изучение приемов работы в CASE-средстве проектирования и моделирования StarUML. Мы выбираем подход Rational Approach для удобства дальнейшей работы.

Сохраните созданный вами проект под любым именем.

4. Диаграмма вариантов использования

Поведение системы (т.е. функциональность, которую она обеспечивает) описывают с помощью функциональной модели, которая отображает системные прецеденты (use cases, случаи использования), системное окружение (действующих лиц, актеров, actors) и связи между ними (use cases diagrams).

Диаграмма вариантов использования (диаграмма прецедентов, use case diagram) — это диаграмма, на которой изображаются отношения между актерами и вариантами использования.

С помощью этой диаграммы можно:

- Определить общие **границы и контекст** моделируемой предметной области на начальных этапах проектирования системы;
- Сформулировать общие **требования** к функциональному поведению проектируемой системы;

- Разработать исходную **концептуальную модель** системы для ее последующей детализации в форме логических и физических моделей;
- Подготовить **исходную документацию** для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Диаграмма вариантов использования (прецедентов) представляет собой граф, в вершинах которого расположены актеры или прецеденты, связи между вершинами – это разного вида отношения.

Актером (действующее лицо, actor) называется любой объект, субъект или система, взаимодействующая с моделируемой системой извне.

Это может быть человек, техническое устройство, программа или любая другая система, которая служит источником воздействия на моделируемую систему так, как определит разработчик. На диаграммах вариантов использования актер изображается в виде человечка, под которым записывается его имя (рис. 8).

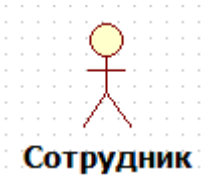


Рисунок 8. Действующее лицо (актер)

Вариант использования (прецедент, use case) — описание множества последовательности действий (включая варианты), выполняемых системой для того, чтобы актер мог получить определенный результат [1].

Каждый вариант использования должен быть независимым в том смысле, что если он всегда выполняется совместно с другим вариантом использования, то, скорее всего, это один прецедент, а не два, либо они связаны отношением включения или расширения, о чем речь пойдет позже. Как следует из определения, прецедент (или вариант использования) должен обладать результирующей ценностью для актера, актер должен получить некоторый результат исполнения прецедента. Скорее всего, после исполнения прецедента в системе произойдут некоторые изменения: появятся новые данные, изменится поведение. Каждый вариант использования должен исполняться от начала до конца.

Прецедент описывает, *что делает* система, но никак *не уточняет, как* она это делает. Заметим, что диаграмма прецедентов не отображает последовательность, в которой будут выполняться варианты использования. На диаграмме прецедент изображается в виде эллипса. Имя прецедента может состоять из нескольких слов и знаков препинания (за исключением

двоеточия), как правило, имя выбирают в виде словосочетания или глагольного выражения (рис. 9).

Рисунок 9. Варианты использования (прецеденты)

Одним из наиболее важных (и дорогостоящих) этапов проектирования информационных систем является этап определения требований к системе. Если требования заказчика информационной системы разработчиками будут определены не корректно, то в итоге заказчик может получить совсем не ту систему, которую он ожидал.

Моделирование прецедентов и актеров помогает нам лучше понять требования, предъявляемые к системе, и согласовать их с заказчиком с помощью демонстрации и обсуждения диаграммы прецедентов. Прецеденты и актеры – это отражение требований к системе, они показывают, кто и для чего будет использовать будущую систему.

Пример. Определим актеров и прецеденты системы заказов магазина «Style».

Напомним, что покупатель делает заказ, складывая товары в корзину. Возможна только одна форма оплаты: банковской картой по интернету, невозможно оформление заказа без оплаты. Заказ имеет статус: оплачен, передан на комплектацию, собран, получен. Статус заказа изменяется автоматически либо сотрудником магазина. Покупатель может узнать статус своего заказа по уникальному номеру заказа.

Система не занимается поставками товаров в магазин. Этим занимается другая система, назовем ее Склад.

Таким образом, с нашей системой взаимодействуют покупатель, сотрудники магазина и внешняя система Склад. С нашей системой взаимодействуют сотрудник отдела продаж, который проверяет оплату заказа и отправляет его на комплектацию, и кладовщик, который собирает заказ и выдает его покупателю. С точки зрения бизнеса – это две разных должности, выполняющих разные функции, но с точки зрения системы они играют одну роль сотрудника, изменяющего статус заказа покупателя с использованием программного обеспечения моделируемой системы. В этом смысле для системы нет разницы между сотрудником отдела продаж и кладовщиком. Выбирая действующих лиц, нужно помнить о том, что мы должны отразить их роль, а не должность. Введем обобщающее сотрудников действующее лицо – Сотрудник. Другой пример: сотрудник магазина «Style» (положим,

кладовщик) может выступать в роли сотрудника и общаться с системой как сотрудник магазина, а может выступать и в роли покупателя, сделав заказ в магазине. Не смотря на то, что физически это один человек, он выступает в роли двух актеров: покупателя и сотрудника. Итак, актеры системы заказов магазина «Style» будут следующие:

Покупатель, Сотрудник, Система Склад.

Покупатель использует нашу систему для того, чтобы заказать вещи, он просматривает каталог, добавляет понравившиеся ему товары в корзину, открывает корзину, удаляет из нее товары или изменяет их количество и, наконец, может оформить свой заказ, при этом его оплатив. В конечном итоге результат использования системы покупателем будет получен, если он выполнил все эти действия от начала до конца. Поэтому не будем разделять заказ товаров на несколько прецедентов, а выделим только один: Заказ товаров.

Покупатель, сделав заказ в магазине «Style», может в дальнейшем узнавать статус своего заказа, это тоже случай использования системы, назовем его Получение информации о заказе.

Сотрудник должен изменять статус сделанного заказа, для него определим прецедент Управление статусом заказа.

Система Склад должна получать информацию о сделанных заказах для возможности управления наличием товаров на складе, для нее также должен быть доступен прецедент Получение информации о заказе.

Итак, прецеденты системы заказов магазина «Style»: Заказ товаров, Управление статусом заказа, Получение информации о заказе.

4.1 Отношения между прецедентами и актерами

Связи и взаимоотношения, существующие между элементами модели, в UML описываются с помощью отношений, изображаемых на диаграммах.

Отношение (relationship) — это семантическая связь между отдельными элементами модели.

Между актерами и прецедентами диаграммы вариантов использования могут существовать разного рода отношения, показывающие, что экземпляры действующих лиц и вариантов использования взаимодействуют друг с другом. Действующие лица могут взаимодействовать с несколькими прецедентами и между собой, равно как и прецеденты могут быть связаны между собой особого типа отношениями.

В основном на диаграммах прецедентов используются следующие типы отношений:

- *ассоциации (association relationship);*
- *включения (include relationship);*
- *расширения (extend relationship);*
- *обобщения (generalization relationship).*

Ассоциация – это структурное отношение, показывающее, что объект неким образом связан с другим объектом.

Отношение этого типа используется не только на диаграммах прецедентов, но и на других диаграммах. Если между элементами модели показано отношение ассоциации, то это означает, что между ними существует семантическая связь. Ассоциативное отношение может быть направленным. В этом случае направление связи показывает, кто является инициатором. Если отношение направленно от актера к прецеденту, то это означает, что актер инициирует выполнение прецедента.

Пример. Покупатель в системе заказов магазина «Style» инициирует выполнение прецедента Заказ товаров (см. рис. 10).

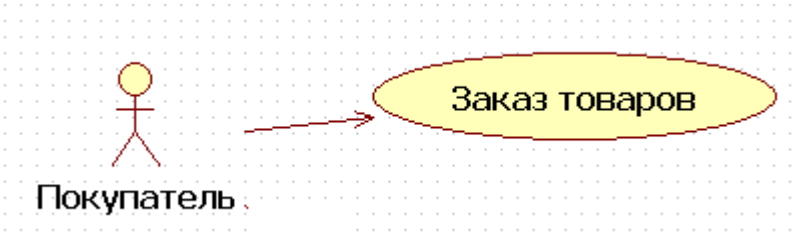


Рисунок 10. Отношение ассоциации между актером и прецедентом

Между прецедентами также возможны взаимоотношения, которые описываются отношениями двух типов: включения и расширения (дополнения).

Включение (include) говорит о том, что исходный прецедент явным образом включает в себя поведение целевого [2].

Другими словами, включение создается, когда один прецедент использует другой. При этом исполнение базового прецедента невозможно без исполнения используемого. Изображается включение в виде пунктирной стрелки с надписью <<include>>, которая направлена от базового элемента к используемому.

Пример. В системе заказов магазина «Style» невозможен заказ товаров без оплаты. На диаграмме прецедентов это можно отразить так, как это показано на рисунке 11.

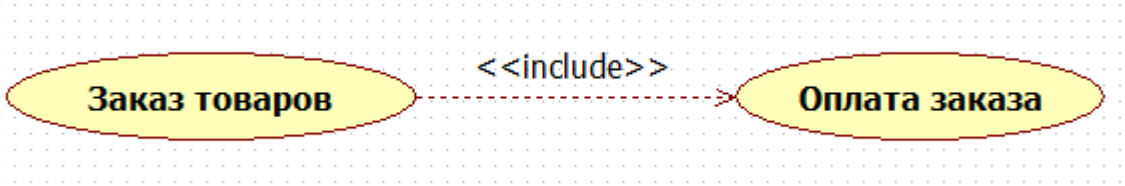


Рисунок 11. Отношение включения между прецедентами

Расширение (extend) показывает, что целевой прецедент расширяет поведение исходного.

Используемый прецедент выполняется не всегда вместе с базовым, а только при выполнении дополнительных условий, таким образом, расширяя функциональность базового элемента. Изображается расширение пунктирной стрелкой с надписью <<extend>>, направленной от используемого варианта использования к базовому.

Пример. При заказе товаров в системе заказов магазина «Style» покупатель может изменить содержание корзины перед тем, как оформить заказ окончательно, а может оставить корзину без изменений. Изменение корзины – это опция, которую на диаграмме вариантов использования мы можем изобразить с помощью расширения (рис. 12).

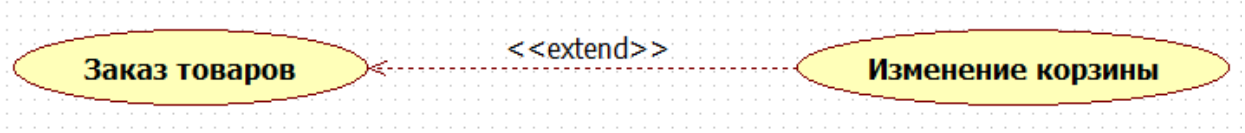


Рисунок 12. Отношение расширения между прецедентами

Обозначения отношений <<include>> и <<extend>> есть не что иное, как обозначения стереотипов, которые широко используются в UML для создания новых элементов модели путем расширения функциональности базовых элементов.

Стереотип (Stereotype) – это механизм, позволяющий категоризировать элементы модели.

С помощью стереотипов мы можем создавать своего рода подтипы типов. Это позволяет UML иметь минимальный набор элементов, которые могут быть дополнены при необходимости для создания связующих базовых элементов в системе. В UML стереотип обозначается именем, которое записывается в двойных угловых скобках: **<<имя стереотипа>>.**

В UML мы можем создавать собственные стереотипы на основе уже имеющихся типов, но также существуют и стандартные, заранее определенные стереотипы нотации UML. Так, отношение зависимости (о котором мы еще будем говорить) расширяется для прецедентов и актеров с помощью двух стереотипов <<include>> и <<extend>>.

Ассоциация – это коммуникативное отношение, которое соответствует стереотипу <<communicate>>, который, впрочем, всегда опускается.

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 отношения обобщения с другим, возможно, абстрактным актером, который моделирует соответствующую общность ролей.

Обобщение (Generalization) – это отношение между общей сущностью и ее конкретным воплощением [2].

На диаграммах обобщение обозначается стрелкой с не закрашенным треугольником на конце, направленной от частного элемента к общему.

Пример. Для изменения статуса заказов в магазине «Style» с проектируемой системой будут работать сотрудник отдела продаж и кладовщик. На диаграмме мы можем показать с помощью отношения обобщения взаимосвязь между актером Сотрудник и актерами Сотрудник отдела продаж и Кладовщик (рис. 13).

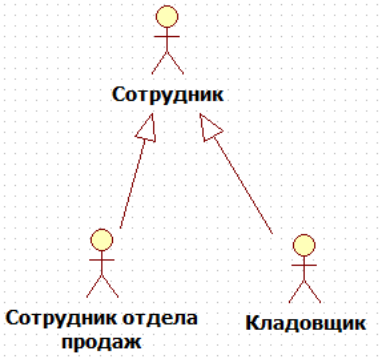


Рисунок 13. Отношение обобщения между актерами

Актеры, прецеденты и отношения – это основные элементы нотации диаграмм вариантов использования. Диаграмма вариантов использования помогает отобразить основные требования к моделируемой системе и обеспечить взаимопонимание функциональности системы между разработчиком и заказчиком. Можно построить одну, главную диаграмму прецедентов, на которой будут отражены границы системы (актеры) и ее основная функциональность (прецеденты). Для более подробного представления системы допускается построение вспомогательных диаграмм прецедентов.

4.2 Построение диаграммы прецедентов в StarUML

В StarUML главная диаграмма прецедентов называется Main и располагается в представлении Use Case. Если в навигаторе модели щелкнуть

два раза по имени этой диаграммы, то откроется ее рабочее поле. Для того чтобы создать прецедент, щелкните по овальному символу прецедента на панели элементов слева от рабочего поля диаграммы, а затем щелкните по тому месту на рабочем поле диаграммы, в которое вы хотите поместить прецедент. Аналогичным образом создается актер. Когда элемент помещается на поле диаграммы, он становится доступен для редактирования имени и некоторых свойств. В выделенное поле введите новое имя прецедента или актера (рис. 14).

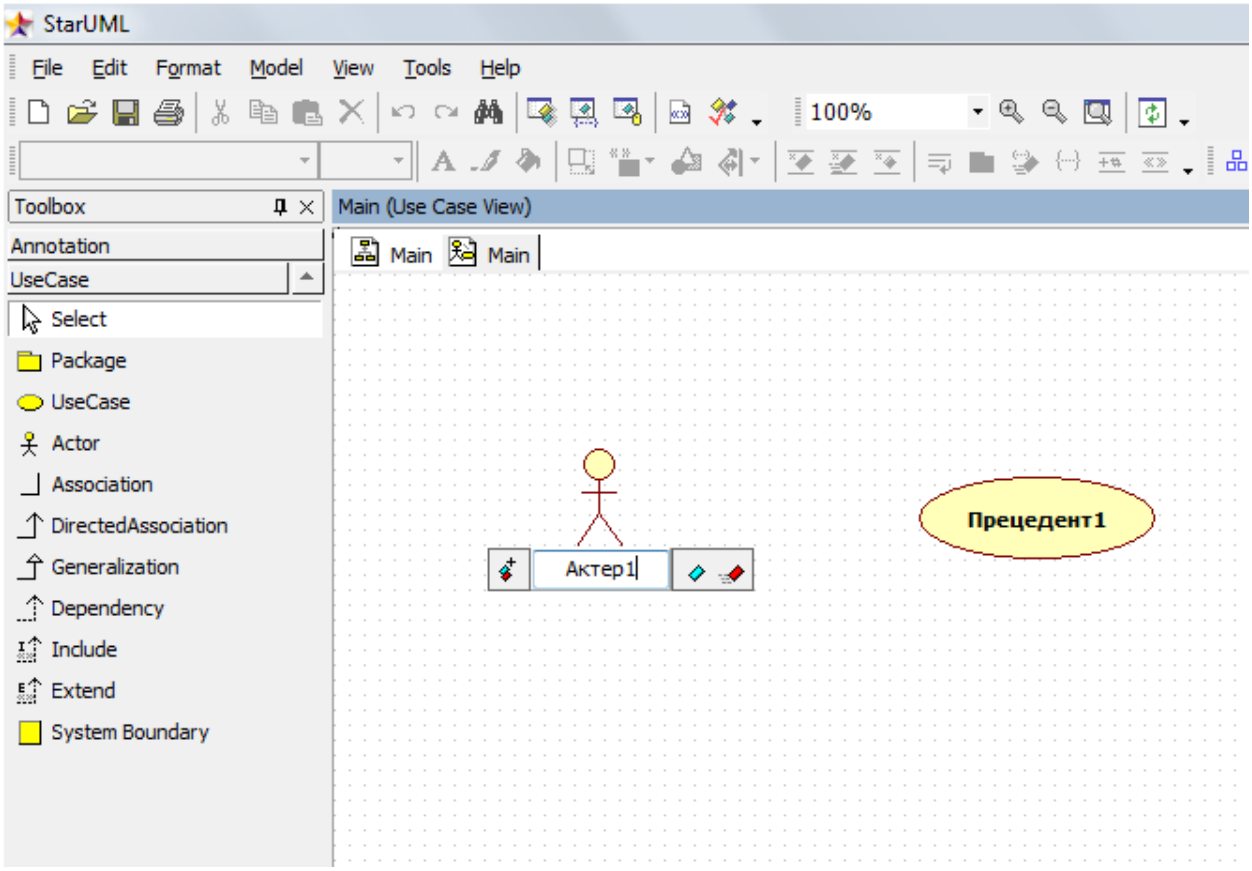


Рисунок 14. Именование элементов в StarUML

Для создания отношения между элементами диаграммы щелкните по изображению соответствующего отношения на панели элементов справа, а затем проведите линию от одного элемента к другому, удерживая левую кнопку мыши (рис. 15).

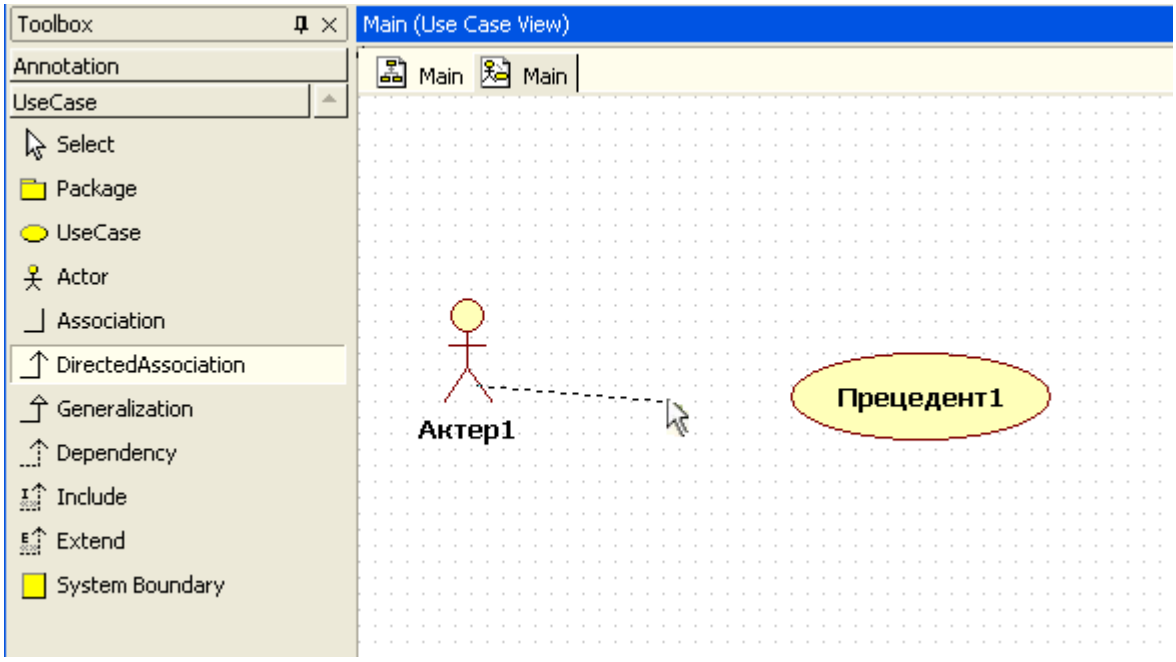


Рисунок 15. Создание отношений между элементами в StarUML

Чтобы удалить элемент с диаграммы достаточно щелкнуть левой кнопкой мыши по этому элементу, а затем нажать кнопку Delete, либо щелкнуть правой кнопкой мыши по элементу и в контекстном меню выбрать Edit -> Delete (рис. 16).

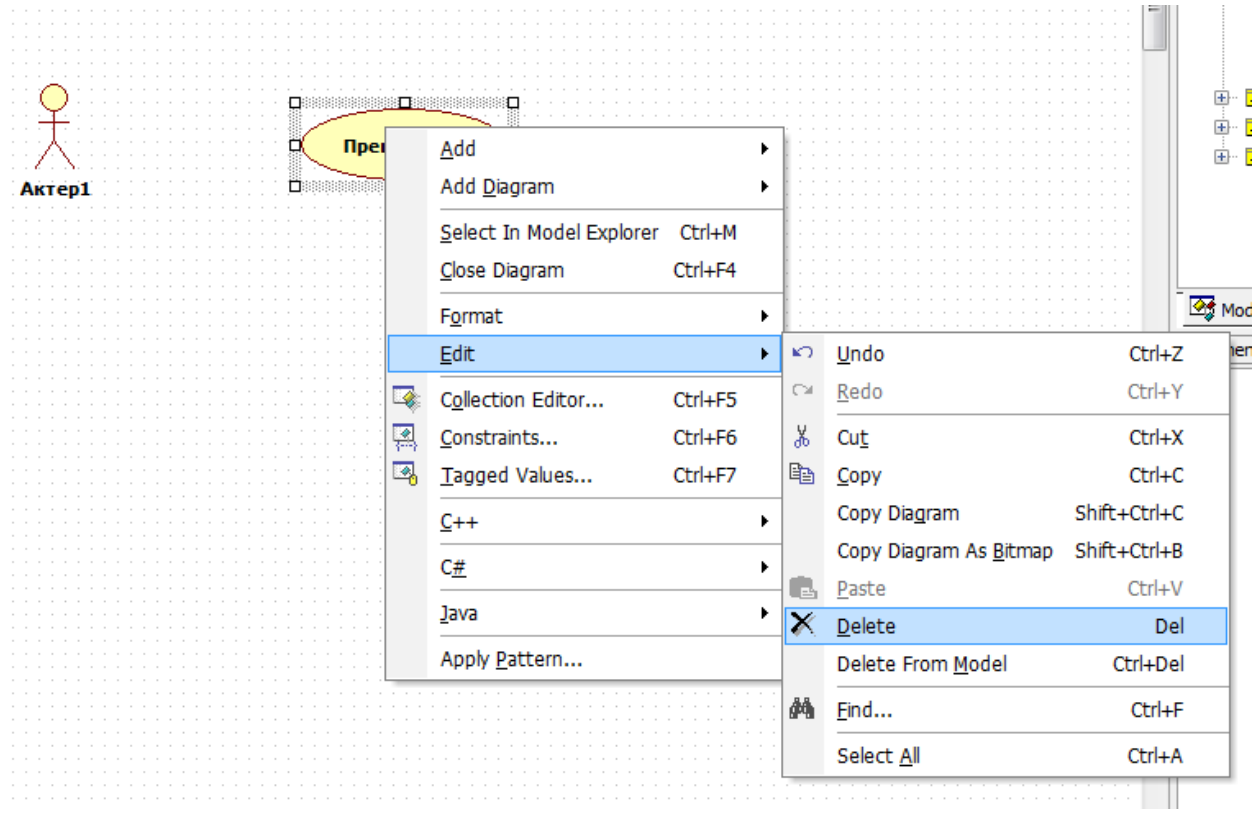


Рисунок 16. Удаление элемента диаграммы в StarUML

Обратите внимание, что элемент был удален с диаграммы, но не из модели (рис. 17)! Его можно найти в навигаторе модели, не смотря на то, что на диаграмме он больше не отображается (элемент Прецедент1):

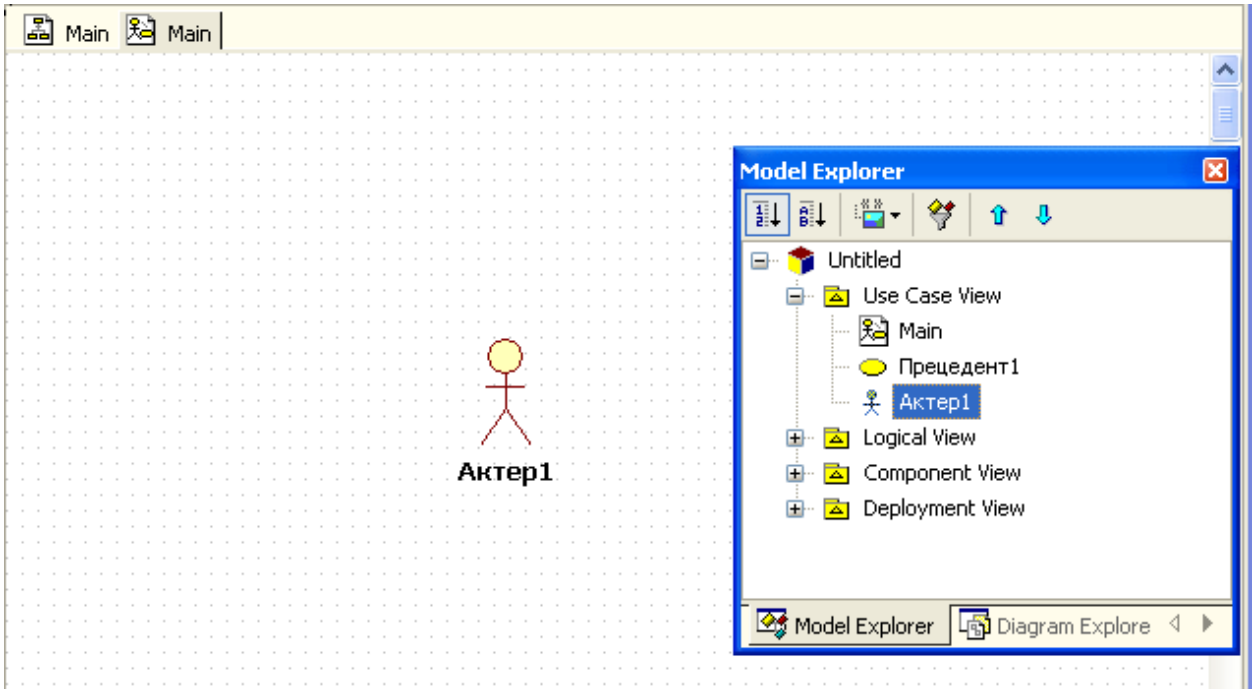


Рисунок 17. Элемент Прецедент1 удален с поля диаграммы, но отображается в навигаторе модели

Если мы передумали и решили вернуть элемент на диаграмму, то это можно сделать, перетащив его с навигатора модели на поле диаграммы.

Для того чтобы **удалить элемент из модели** нужно щелкнуть по нему на диаграмме или по его изображению в навигаторе модели правой кнопкой мыши и в контекстном меню выбрать пункт Delete from Model. Элемент будет полностью удален.

Описанные выше способы добавления и удаления элементов и отношений могут быть использованы для построения диаграмм любых типов. Мы не будем в дальнейшем заострять на этом внимание читателя. Заметим, что все описанные операции доступны также из главного меню StarUML.

Пример. Для системы заказов магазина «Style» мы определили актеров Покупатель, Сотрудник, Система Склад и прецеденты Заказ товаров, Управление статусом заказа, Получение информации о заказе. Построим основную диаграмму прецедентов (рис. 18).

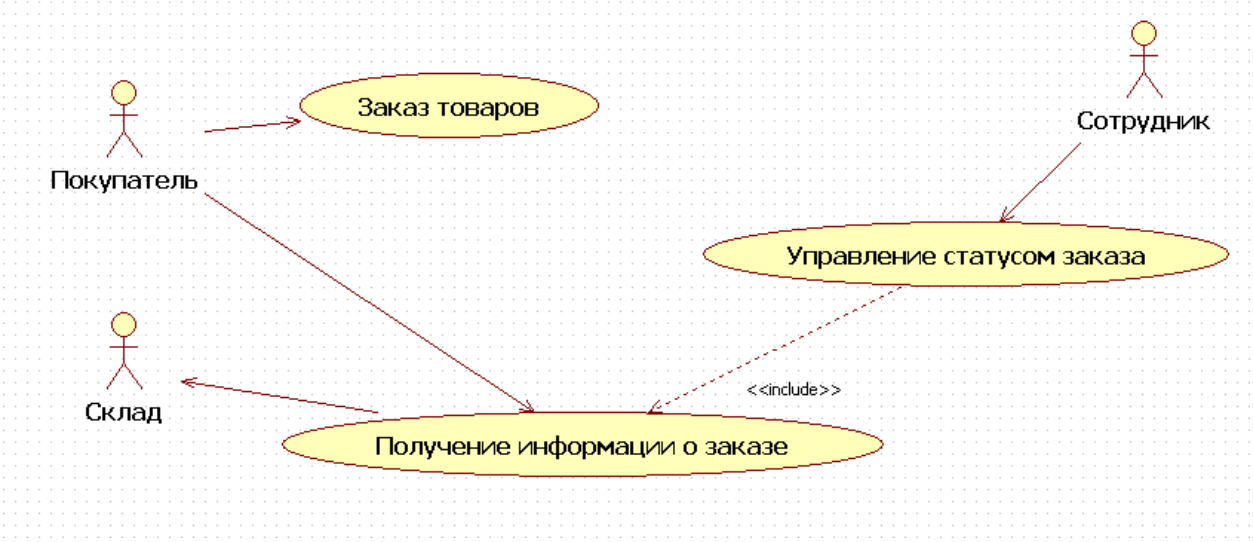


Рисунок 18. Основная диаграмма вариантов использования системы заказов магазина "Style"

Для актера Покупатель и прецедента Заказ товаров установили отношение направленной ассоциации: Заказ товаров инициализируется Покупателем. Сотрудник имеет возможность управлять статусом заказа, при этом он непременно участвует в прецеденте Получение информации о заказе. Направленную ассоциацию от Получение информации о заказе к актеру Система Склад можно понимать как автоматическую передачу данных из моделируемой системы в систему снабжения товарами Склад.

В модель нужно включить краткое описание каждого актера или прецедента, делается это для того, чтобы между разработчиком и заказчиком системы не оставалось «белых пятен» и расхождений в понимании функциональности системы и ролей взаимодействующих с ней актеров. Для каждого актера описывается роль, которую он играет в системе, а для каждого прецедента – его назначение и функциональность. Также можно уточнить, каким актером запускается прецедент.

4.3 Документирование элементов модели в StarUML

В StarUML добавление описания к элементам модели делается следующим образом. Выделите элемент модели, щелкнув по нему мышкой, и откройте редактор Documentation. Если он не отображается справа на одной из вкладок инспектора модели, то откройте его, используя меню View → Documentation. Напротив пункта Documentation должна стоять галочка. Введите описание элемента в окно документирования (рис. 19).

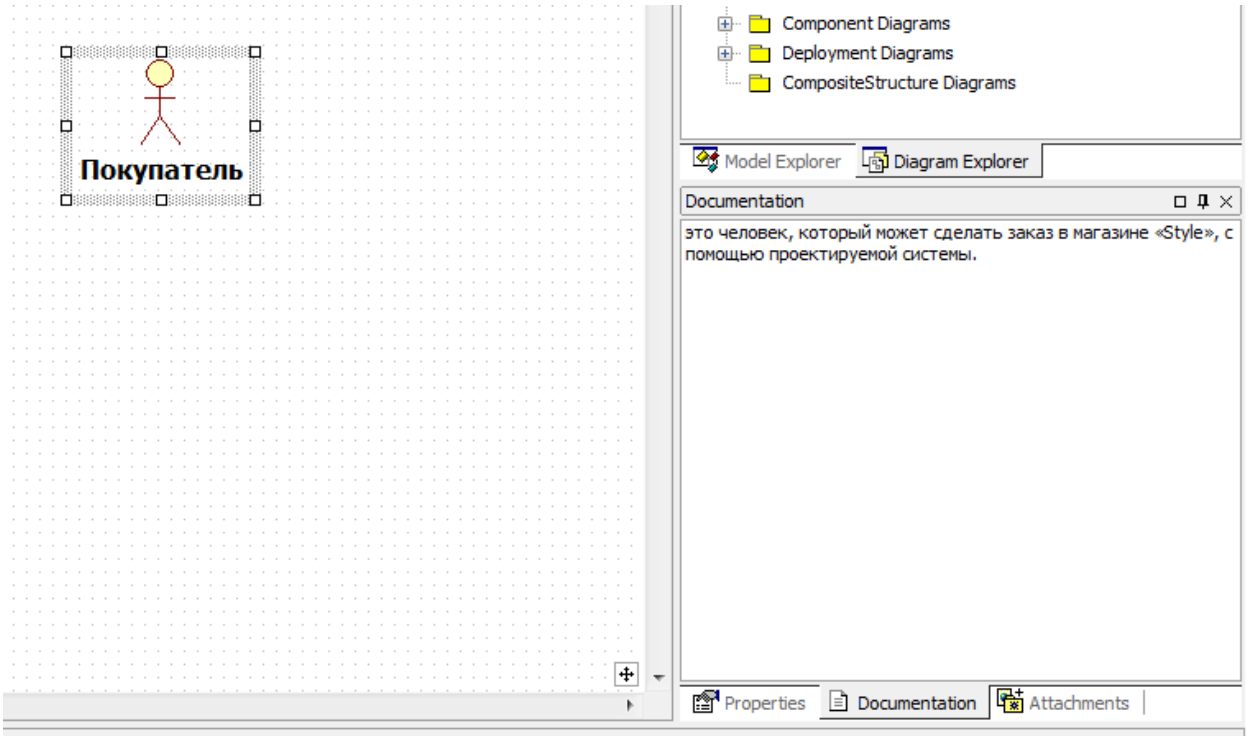


Рисунок 19. Документирование элемента модели в StarUML

Все элементы модели должны быть задокументированы. Описанный выше способ подходит для любого элемента любой диаграммы.

Пример. Для актеров и прецедентов системы заказов магазина «Style» сделаем краткое описание.

Покупатель – это человек, который может сделать заказ в магазине «Style», с помощью проектируемой системы.

Сотрудник – это все сотрудники магазина «Style», которые могут получать информацию о сделанных заказах и изменять статус заказа в системе в зависимости от того шага, на котором находится обработка данного заказа.

Система Склад – это внешняя система, которая получает информацию о сделанных в магазине «Style» заказах для того, чтобы обеспечить учет наличия товаров на складе и снабжение товарами.

Заказ товаров – этот прецедент запускается покупателем для того, чтобы оформить заказ в магазине «Style». Состоит из просмотра каталога, добавления товаров в корзину, просмотра корзины, изменения содержания корзины и оформления заказа, включая оплату.

Управление статусом заказа – этот прецедент используется сотрудниками магазина для изменения статуса заказа в процессе его обработки.

Получение информации о заказе – прецедент используется всеми актерами для просмотра информации о заказе.

Для того чтобы создать еще одну диаграмму (любого типа), например, детализирующую прецедент, щелкните правой кнопкой мыши по папке Use Case Model и в появившемся контекстном меню выберите Add Diagram, затем выберите из списка диаграмму, которую вы хотите добавить. Например, можно создать дополнительную диаграмму прецедентов, выбрав пункт Use Case Diagram (рис. 20).

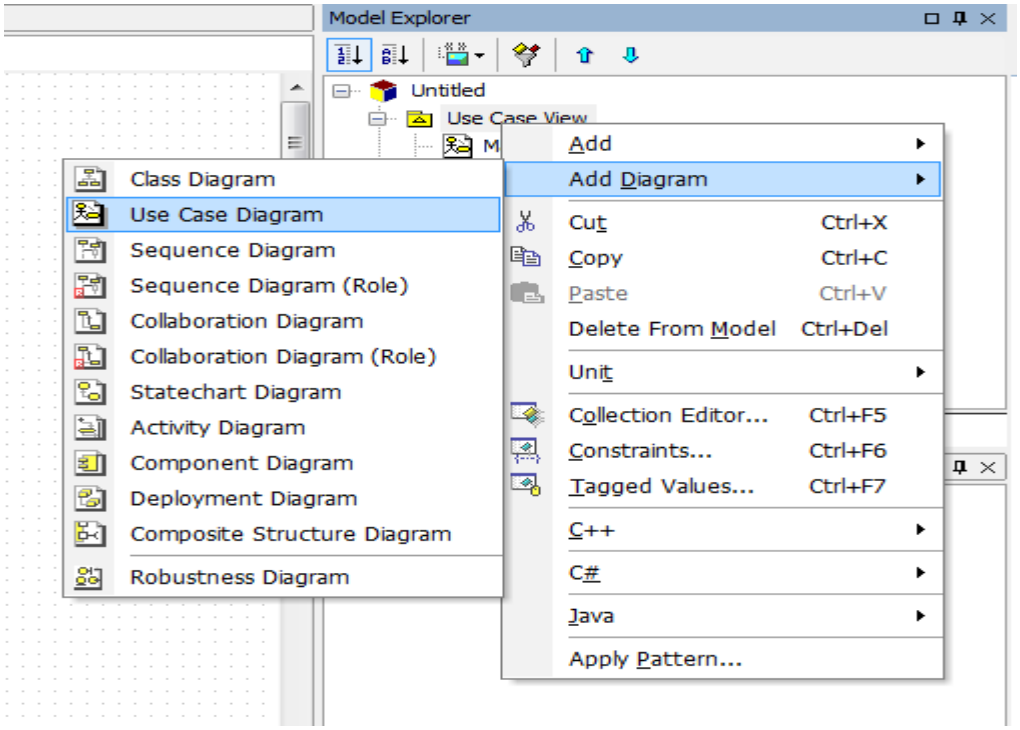


Рисунок 20. Создание дополнительной диаграммы прецедентов

Пример. Наиболее значимым для данной системы и ее актеров прецедентом является прецедент Заказ товаров. Для него мы построим дополнительную диаграмму прецедентов, поясняющую этот вариант использования (рис. 21).

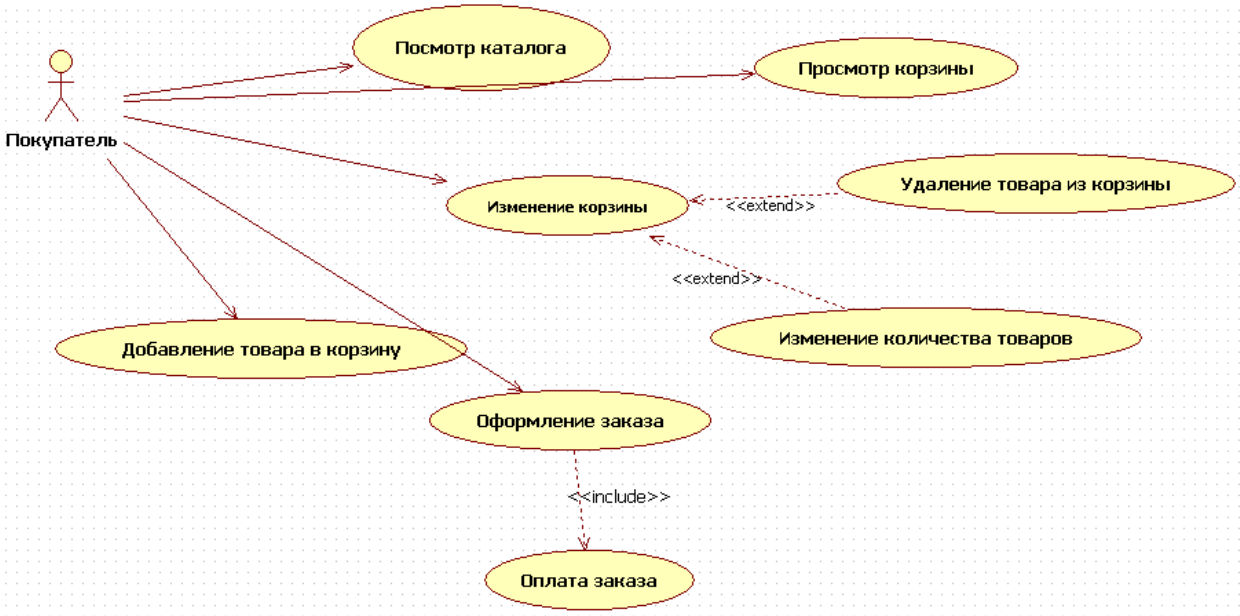


Рисунок 21. Диаграмма вариантов использования, поясняющая основной прецедент Заказ товаров

5. Потоки событий

Одним из требований языка UML является самодостаточность диаграмм для представления информации о моделях проектируемых систем. Однако, как уже отмечалось выше, диаграммы вариантов использования описывают то, что делает система, без уточнения того, как она это делает.

Для реального описания системы потребуются более специфические данные, которые отражены в потоке событий. Потоки событий уточняют или детализируют последовательность действий, совершаемых системой при выполнении ее вариантов использования, а также описывают логику переходов через варианты использования.

Поток событий – это определенная последовательность действий, которая описывает действия актеров и поведение моделируемой системы в форме обычного текста.

Потоки событий – это текстовые описания пошагового выполнения прецедентов, они понятны не только разработчику, но и стороннему читателю. Их задача – еще больше детализировать описание функциональности системы до того, как разработчики приступят к написанию программного кода, и устранить возможное недопонимание требуемой функциональности, как можно больше сблизить представления разработчика о системе и заказчика.

Потоки событий бывают трех типов: основной, альтернативный и поток ошибок.

Основной (главный) поток описывает наилучший сценарий либо наиболее используемый путь исполнения прецедента.

Альтернативный поток специфицирует отклонения от основного потока, которые не рассматриваются как ошибочные.

Поток ошибок рассматривается как отклонение от альтернативного или основного, которое порождает условия формирования ошибки.

Пример. Опишем потоки событий прецедента Заказ товаров.

Основной поток событий.

1. Прецедент начинается с выбора покупателем режима показа каталога.
2. Система открывает каталог.
3. Покупатель выбирает режим показа корзины.

A1. Покупатель просматривает каталог и запускает поток «добавление товара в корзину»

4. Система открывает корзину.
5. Покупатель нажимает кнопку «оформить заказ».

A2. Покупатель просматривает корзину и запускает поток «изменение корзины».

A3. Покупатель решает вернуться в каталог.

6. Система переходит к первому шагу оформления заказа: запрашивает у покупателя личные данные и телефон.
7. Покупатель вводит личные данные и телефон.
8. Система переходит ко второму шагу оформления заказа: показывает содержимое заказа и просит подтвердить заказ.
9. Покупатель подтверждает заказ.

A4. Покупатель возвращается в корзину.

10. Система переходит к третьему шагу оформления заказа: запрашивает тип кредитной карты, ее номер, секретный код, имя владельца и дату завершения срока действия.
11. Покупатель вводит тип кредитной карты, ее номер, секретный код, имя владельца и дату завершения срока действия.
12. Система переходит к четвертому шагу оформления заказа: подтверждает оплату.

A5. Счет пользователя не найден.

A6. Недостаточно денег на счете.

E1. Платежная система недоступна.

13. Система присваивает заказу номер и отправляет его вместе с подтверждением заказа на электронный адрес покупателя.

14. Вариант использования завершается.

Альтернативные потоки.

А1. добавление товара в корзину

1. Покупатель выбирает размер.
2. Покупатель выбирает количество.
3. Покупатель нажимает кнопку Добавить в корзину
4. Система помещает выбранный товар в корзину.

А7. Покупатель не выбрал размер.

А8. Покупатель не выбрал количество.

5. Система выводит сообщение о том, что товар добавлен в корзину.
6. Поток возвращается на второй этап основного потока.

А2. Изменение корзины

1. Покупатель нажимает кнопку «Удалить» напротив одного выбранного товара.

А9. Покупатель изменяет количество позиций одного выбранного товара.

2. Система удаляет товар из корзины.
3. Поток возвращается к этапу 4 основного потока.

А3. Покупатель решает вернуться в каталог

1. Поток возвращается к этапу 2 основного потока.

А4. Покупатель возвращается в корзину

1. Поток возвращается к этапу 4 основного потока.

А5. Счет пользователя не найден

1. Система выводит сообщение о том, что счет пользователя не обнаружен.
2. Поток возвращается к этапу 11 основного потока.

А6. Недостаточно денег на счете

1. Система выводит сообщение о том, что на счете пользователя недостаточно денег для совершения операции.
2. Поток возвращается к этапу 11 основного потока.

А7. Покупатель не выбрал размер

1. Система выводит сообщение о необходимости выбора размера.
2. Поток возвращается на 1 этап потока А1.

А8. Покупатель не выбрал количество

1. Система выводит сообщение о необходимости выбора количества товара.

2. Поток возвращается на 2 этап потока A1.

A9. Покупатель изменяет количество позиций одного выбранного товара

1. Покупатель увеличивает количество товара.
2. Система обновляет корзину.

A10. Недостаточно товара в наличии.

3. Поток возвращается к этапу 4 основного потока.

A10. Недостаточно товара в наличии

1. Система выводит сообщение о том, сколько позиций может заказать покупатель.
2. Поток возвращается к этапу 4 основного потока.

Потоки ошибок.

E1. Платежная система недоступна

1. Система выводит сообщение о недоступности платежной системы.
2. Поток возвращается к этапу 11 основного потока.

5.1 Добавление потока событий к модели в StarUML

Чтобы добавить поток событий к модели, нужно выделить прецедент, который он детализирует (в данном примере это прецедент Заказ товаров), в инспекторе модели открыть редактор вложений Attachments, нажать на значок  , расположенный в верхней части редактора вложений, в появившемся окне нажать кнопку  и выбрать соответствующий файл, содержащий описание потока событий. Чтобы удалить вложенный файл, выделите его на редакторе вложений и нажмите кнопку  (рис. 22).

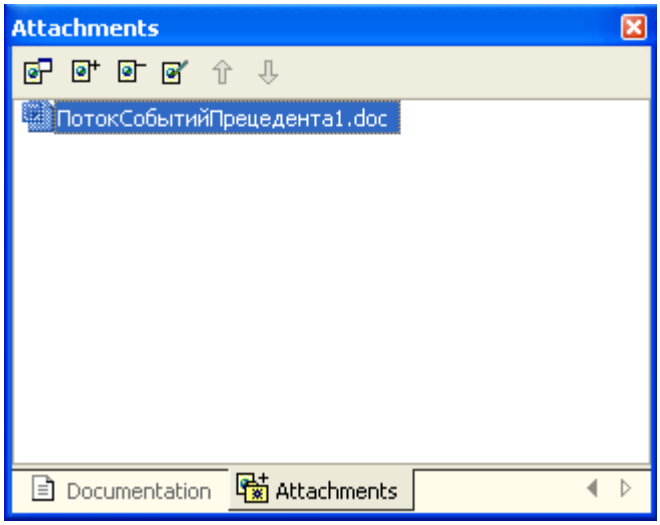


Рисунок 22. Добавление вложений в модель

6. Диаграммы деятельности

Диаграммы деятельности обеспечивают еще один способ моделирования потока событий. С помощью текстового описания можно рассказать о потоке, но трудно будет понять логику событий в сложных и запутанных потоках с множеством альтернативных ветвей.

Диаграммы деятельности создаются также на разных этапах жизненного цикла системы для отражения последовательности выполнения операций.

6.1 Основные элементы нотации диаграмм деятельности

Рассмотрим основные элементы нотации диаграмм деятельностей. На них иллюстрируются деятельности, переходы между ними, элементы выбора и синхронизации.

Деятельностью называется исполнение определенного поведения в потоке управления системы. В UML деятельность изображается в виде скругленного прямоугольника с текстовым описанием внутри.

Пример. Деятельность обозначает некоторый шаг (этап) процесса. В прецеденте Заказ товаров одним из таких шагов может быть Добавить товар в корзину (рис. 23).

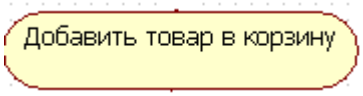


Рисунок 23. Деятельность

Переход показывает, как поток управления переходит от одной деятельности к другой. Обычно переход осуществляется по завершении деятельности (рис. 24).

Пример. В нашем примере выполняя Заказ товаров покупатель может Открыть корзину и Удалить товар из нее. Это две разные деятельности, переход к удалению товара возможен только после открытия корзины.

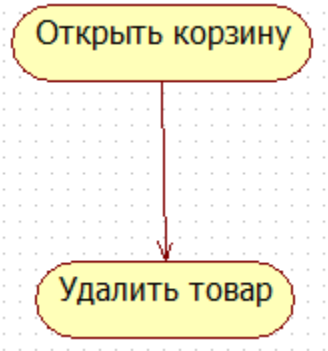


Рисунок 24. Переход между деятельностями

Два состояния на диаграмме деятельности - ***начальное и конечное*** - определяют продолжительность потока. Начальное состояние обязательно должно быть отмечено на диаграмме, оно определяет начало потока. Конечных состояний может быть несколько или не одного. Оно определяет точку завершения потока. Конечных состояний может быть несколько, но начальное должно быть только одно. Начальное состояние изображается жирной точкой, а конечное – жирной точкой в окружности (рис. 25).

Рисунок 25. Обозначения начального и конечного состояний

При моделировании управляющих потоков системы часто бывает необходимо показать места их разделения на основе ***условного выбора***. Выбор на диаграмме показывается ромбом, помещенным на переходе. Ограничительные условия, от которых зависит выбор направления перехода, помещаются обычно над ромбом. В нотации UML условия записываются в квадратных скобках: [условие].

Пример. Если все товары, которые хочет заказать покупатель, добавлены в корзину, то покупатель может просмотреть корзину и оформить заказ. Условие перехода от деятельности Добавить товар в корзину к Просмотреть корзину на диаграмме можно показать так, как это изображено на рисунке 26.

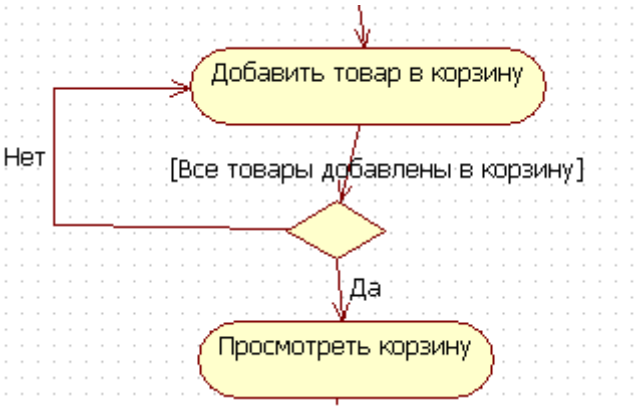


Рисунок 26. Условие перехода между деятельностями

Синхронизация - Это способ показать, что две или более ветвей потока выполняются параллельно.

Деятельности, помещенные между двумя жирными линиями на диаграмме деятельности, исполняются синхронно, одновременно.

Пример. После оплаты заказа покупателем система присваивает заказу уникальный номер и отсылает подтверждение заказа на электронную

почту покупателя. Эти две деятельности можно выполнить синхронно. Как это изображается на диаграмме, показано на рисунке 27.

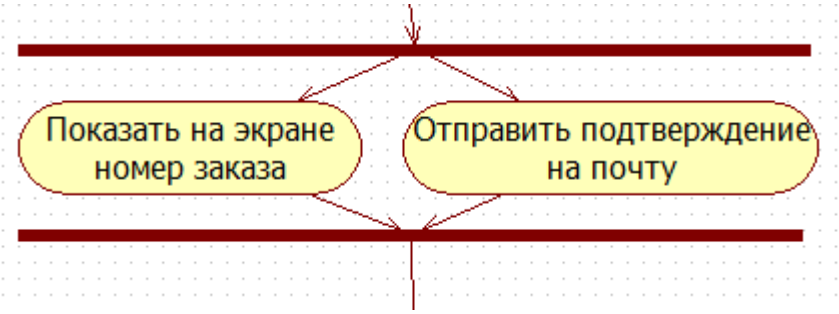


Рисунок 27. Линии синхронизации

Секции делят диаграмму деятельности на несколько участков. Это нужно для того, чтобы показать, кто отвечает за выполнение деятельности и в каком порядке. Если деятельность находится на секции с именем Покупатель, то этот актер и выполняет ее.

Пример. Секция актера Покупатель изображена на рисунке 28.

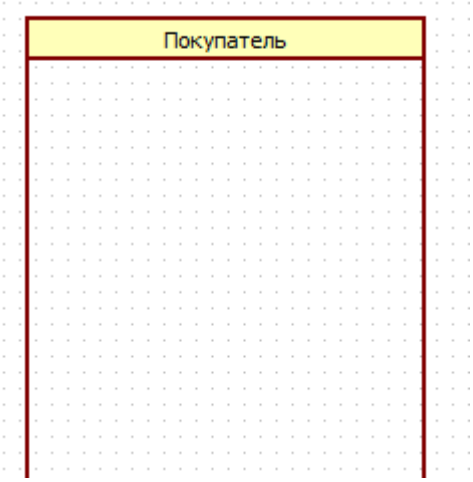


Рисунок 28. Секция

6.2 Создание диаграммы деятельности в StarUML

Чтобы построить диаграмму деятельности для некоторого прецедента в StarUML, нужно щелкнуть правой кнопкой мыши по этому прецеденту, в выпавшем контекстном меню выбрать пункт Add Diagram, затем в появившемся списке выбрать Activity Diagram (рис. 29).

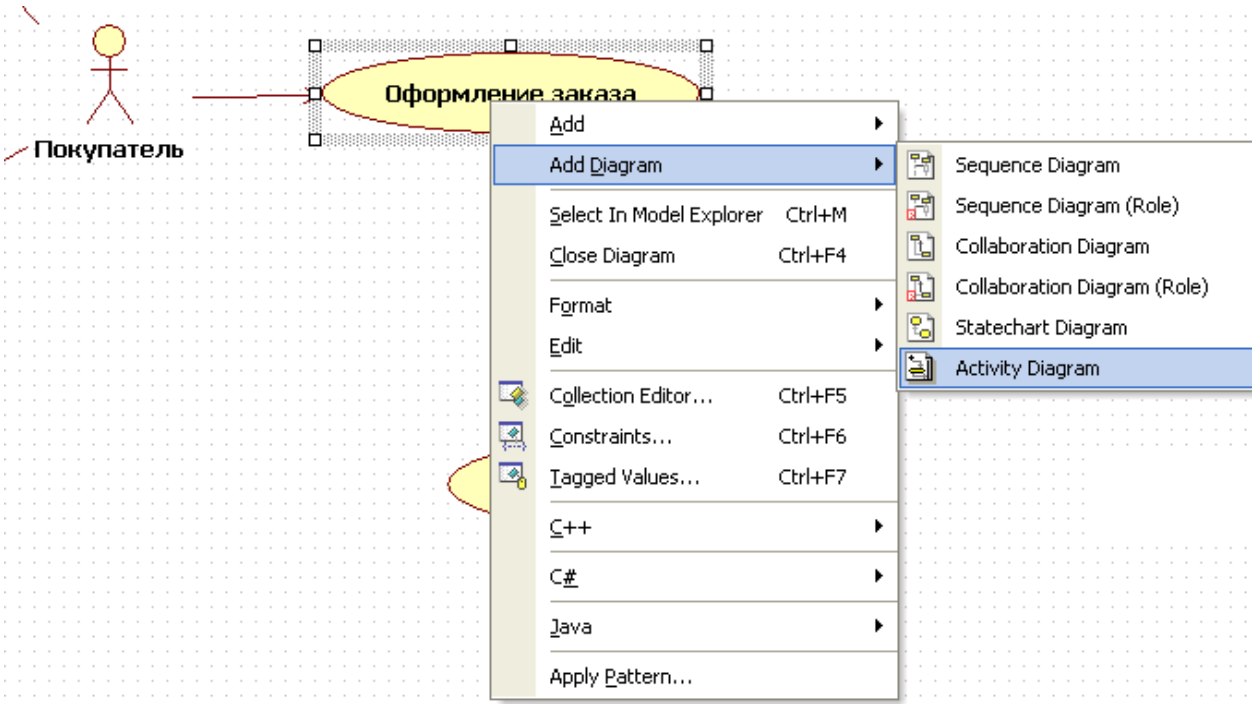


Рисунок 29. Добавление диаграммы деятельности

Поле для создания диаграммы деятельности появится в окне программы, изменится панель инструментов слева, и новая диаграмма отобразится на навигаторе модели.

Пример. Построим диаграмму деятельности для дополнительного прецедента Оформить заказ актера Покупатель (см. рис. 30). Оформление заказа включает указание своих личных контактных данных, электронной почты и оплату заказа. Оформление начинается из корзины покупателя, когда он выбирает опцию «Оформить заказ».

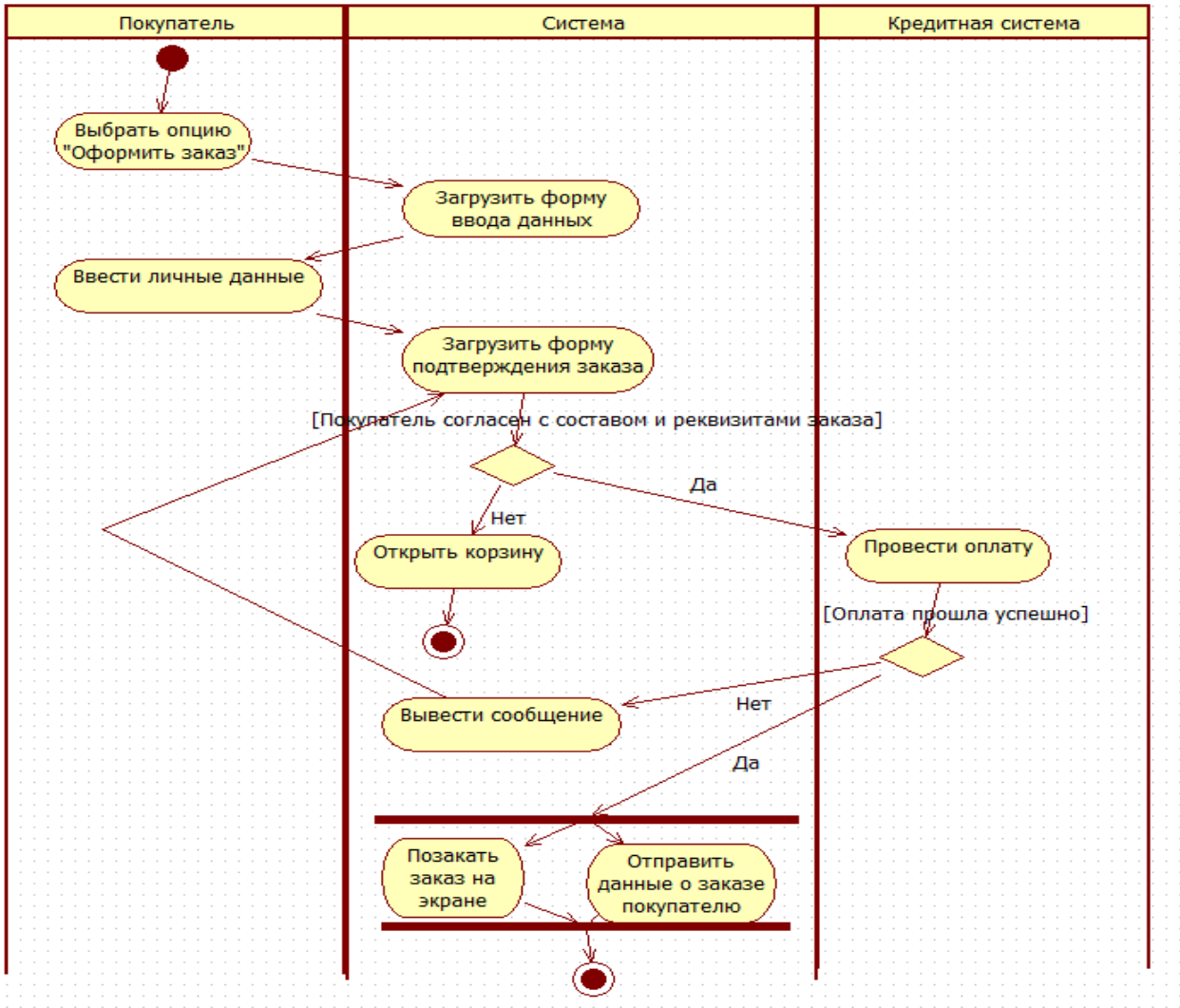


Рисунок 30. Диаграмма деятельности прецедента Оформить заказ

6. Диаграммы классов

Диаграмма классов является частью логической модели системы и представляет статическую картину системы.

Для каждой системы строится не одна, а несколько диаграмм классов: возможно, что для каждого прецедента или сценария своя. На одних показывают подмножества классов, объединенные в пакеты, и отношения между ними, на других – отображают те же подмножества, но с атрибутами и операциями классов. Для представления системы разрабатывается столько диаграмм классов, сколько потребуется.

6.1 Основные элементы диаграмм классов

Дадим некоторые определения и опишем основные элементы нотации диаграмм классов.

Объект – это некоторая сущность реального мира или концептуальная

(абстрактная) сущность.

Пример. Примерами объектов могут служить дом №4 по улице Садовая, сотрудник фирмы Иван Петров, ваш компьютер. Или нечто абстрактное: химическая формула, торговый заказ номер 456789, банковский счет клиента Петра Иванова.

Объект имеет четко определенные границы и значение для системы и характеризуется состоянием, поведением и индивидуальностью.

Состояние объекта – это одно из условий, в котором он может находиться. Состояние обычно изменяется со временем и характеризуется набором свойств, которые называются атрибутами.

Пример. Покупатель определяется его именем, адресом, телефоном, датой рождения.

Поведение определяет, как объект реагирует на запросы других объектов и что может делать сам объект. Поведение характеризуется операциями объекта.

Пример. Покупатель может добавить товар в корзину, просматривать каталог, удалять товар из корзины.

Индивидуальность означает, что каждый объект уникален, даже если его состояние идентично состоянию другого объекта.

Пример. Объекты Мария Петрова и Анна Седова уникальны, хотя каждый из них является покупателем магазина и имеет одинаковые поведение и состояния.

Как правило, в системе существует множество объектов имеющих одинаковое поведение, принимающих одинаковые состояния. Например, сотрудники фирмы, которых может быть несколько десятков, и данные о которых содержатся в базе данных, имеют одинаковые атрибуты – фамилию, имя, отчество, дату рождения, должность и др. – с разными значениями этих атрибутов, а также могут иметь схожее поведение – подать заявление на отпуск или перевод в другое подразделение. Для группировки объектов используются классы.

Класс – это описание группы объектов с общими свойствами (атрибутами), поведением (операциями), отношениями с другими объектами и семантикой.

Каждый класс является шаблоном для создания объекта. А каждый объект – это экземпляр класса. Важно помнить, что ***каждый объект может быть экземпляром только одного класса!***

Пример. Применительно к магазину «Style» мы можем сгруппировать сотрудников магазина, описав общий для них класс Сотрудник. Объект этого класса, например, Иван Петров, может включать в себя следующую информацию: имя, адрес, должность, размер заработной платы, кроме того этот объект может выйти в отпуск.

В нотации UML классы и объекты изображаются в виде прямоугольников (см. рис. 31). Прямоугольник класса всегда делится на три секции (раздела), имя класса помещается в первую секцию, каждое слово в названии класса принято писать с большой буквы. Во второй и третьей секциях могут указываться атрибуты и операции класса соответственно, эти секции могут быть пустыми. Названия классов выбираются в соответствии с понятиями предметной области. Это должно быть существительное или словосочетание в единственном числе, наиболее точно характеризующее предмет. Класс должен описывать только одну сущность.

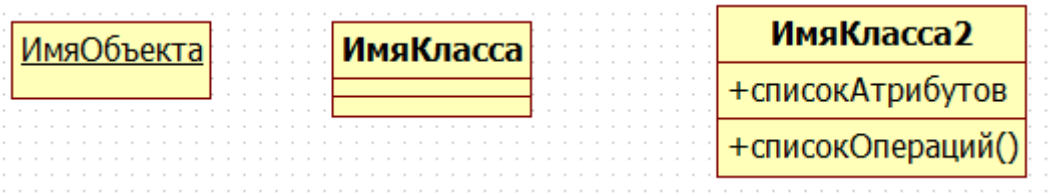


Рисунок 31. Изображение классов и объектов

Имя класса может быть простым, как это показано на рисунке 31, или составным (см. рис. 32). Составное имя класса состоит из самого имени класса и из имени пакета, которому принадлежит класс, разделенных двоеточием. Имя класса должно быть уникальным внутри пакета.

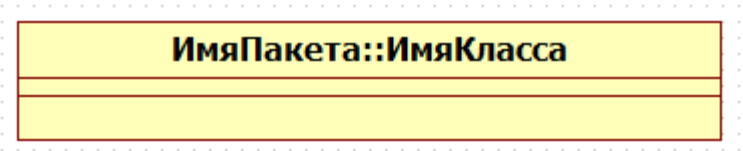


Рисунок 32. Составное имя класса

Составное имя объекта также состоит из имени объекта и имени класса, разделенных двоеточием. Объект может быть анонимным, если неизвестно его настоящее имя. Тогда на диаграмме объект изображается с именем, которое состоит из двоеточия и имени класса, которому принадлежит объект. Если пока неизвестен класс, экземпляром которого является объект, то изображается имя объекта после которого идет двоеточие. Такой объект называется «сиротой» (см. рис. 33).

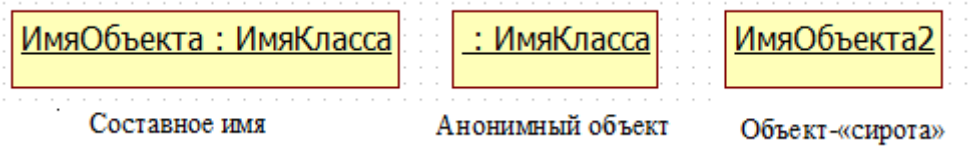


Рисунок 33. Именование объектов

Пример. Класс Сотрудник и объект этого класса - некоторого сотрудника - можно изобразить так, как показано на рисунке 34.

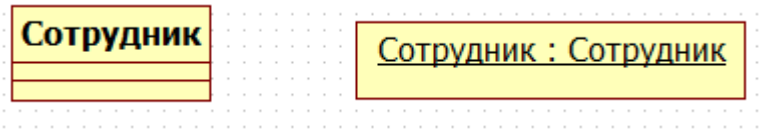


Рисунок 34. Класс и его объект

Мы дали объекту класса Сотрудник имя, совпадающее с именем класса.

6.2 Выявление классов

Выявление классов можно начать с изучения потока событий. Имена существительные в описании этого потока дадут понять, что может являться классом. В общем случае существительное может оказаться действующим лицом, классом, атрибутом класса или выражением, не являющимся ни действующим лицом, ни классом, ни атрибутом класса.

Если в ходе проектирования системы Вы уже построили диаграммы взаимодействия, перед тем, как приступать к построению диаграмм классов, то ищите на этих диаграммах похожие объекты. Например, у Вас может быть диаграмма последовательности, описывающая оформление заказа объектами Ивановым и Петровым. Обратите внимание на эти объекты: они имеют одинаковые свойства: имя, счет в банке и т.п. Значит, в системе должен появиться класс с именем Покупатель, который будет шаблоном объектов Иванов и Петров.

Некоторые возможные классы будут выявлены при рассмотрении трех стереотипов: сущность (entity), граница (boundary) и управление (control). Мы уже встречались со стереотипами отношений, когда говорили об отношениях на диаграммах прецедентов. Тот же принцип создания нового типа на основе уже существующего применим и для классов.

Стереотип – это механизм, позволяющий категоризировать классы. Он используется для создания нового типа элемента, в данном случае нового типа класса.

Например, Вы хотите выделить все экранные формы в модели. Для этого нужно создать стереотип Form (Форма).

Стереотипы помогают лучше понять ответственности каждого класса в

модели, категоризировать выполняемые ими функции. В UML для этого применяют три основных стандартных вида стереотипов классов: классы-сущности, граничные классы и управляющие классы.

Класс-сущность содержит информацию, хранимую постоянно. Используется для моделирования данных и поведения с длинным жизненным циклом. Они могут представлять информацию о предметной области, а могут представлять элементы самой системы. Часто являясь абстракциями предметной области, они имеют наибольшее значение для пользователя, поэтому в их названиях применяются термины предметной области. Если существует проект базы данных, то можно обратиться к изучению названий таблиц, многие из них станут классами-сущностями. Обозначаются классы-сущности стереотипом <<entity>> либо специальной пиктограммой (рис. 35).

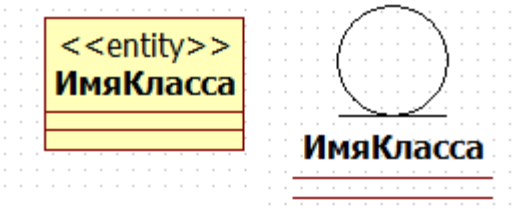


Рисунок 35. Обозначение классов-сущностей

Граничными классами называются классы, расположенные на границе системы со всем остальным миром, и т.о. они обеспечивают взаимодействие между окружающей средой и внутренними элементами системы.

Для вычисления пограничных классов необходимо исследовать диаграммы вариантов использования. Для каждого взаимодействия между актером и прецедентом нужно создать хотя бы один граничный класс. Обратите внимание, что если два действующих лица инициируют один прецедент, то они могут применять один общий пограничный класс для взаимодействия с системой. Обозначаются граничные классы именем стереотипа <<boundary>> либо специальной пиктограммой (рис. 36).

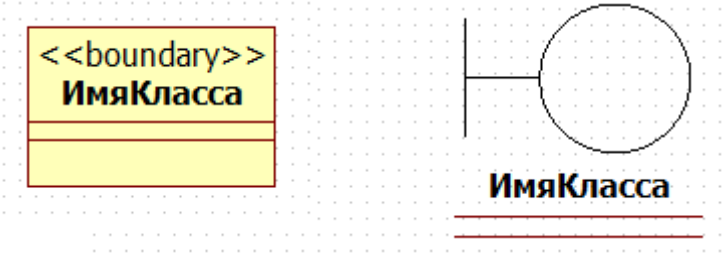


Рисунок 36. Обозначение граничных классов

Управляющий класс отвечает за координацию действий других классов. Они служат для моделирования последовательного поведения одного или нескольких прецедентов и координации событий, реализующих заложенное в них поведение. Обозначаются управляющие классы именем

стереотипа <<control>> либо специальной пиктограммой (рис. 37).

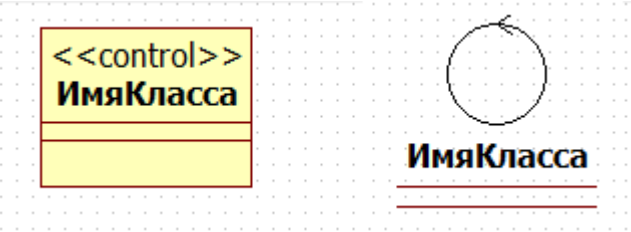


Рисунок 37. Обозначение управляющих классов

Управляющие классы можно представить, как «исполняющие» прецедент, поэтому у каждого варианта использования обычно имеется один управляющий класс, контролирующий последовательность событий этого прецедента. Они обычно зависят от приложения.

Управляющий класс делегирует ответственности другим классам. Сам он может получать мало сообщений, но отсылать множество. Его называют классом-менеджером. Он запускает альтернативные потоки и знает, как поступить в случае ошибки. На начальном этапе проектирования управляющие классы создаются для каждой пары актер/прецедент, в дальнейшем они могут объединяться, разделяться или исключаться.

6.3 Документирование классов

После того, как класс создан, информацию о нем необходимо документировать. Заметим, что документация предназначена для описания предназначения класса, а не его структуры.

Пример. Если в нашей модели присутствует класс Сотрудник, то хорошим описанием для него будет:

Сотрудник – это человек, работающий на фирме. Класс содержит информацию, необходимую для исполнения организацией своих обязанностей по отношению к сотруднику (начисление зарплаты, перевод на другую должность, увольнение и т.п.)

Плохим описанием будет описание структуры класса, которая может быть и так описана с помощью атрибутов. Например, плохое описание класса Сотрудник:

Имя, телефон, адрес, должность, зарплата.

В StarUML документирование классов выполняется также как и описанное выше документирование прецедентов. Нужно выделить класс, который вы хотите описать, открыть окно документирования Documentation на инспекторе модели и ввести описание класса.

6.4 Построение диаграммы классов в StarUML

Диаграммы классов относятся к логическому представлению системы Logical View. На диаграмме Main представления Logical View обычно размещают главную диаграмму пакетов, а диаграммы классов помещают на другие листы этого представления. Для создания новой диаграммы классов выполним следующие шаги: щелкнуть правой кнопкой мыши по папке представления Logical View в навигаторе модели, в контекстном меню выбрать пункт Add Diagram, в списке выбрать диаграмму классов Class Diagram (рис. 38).

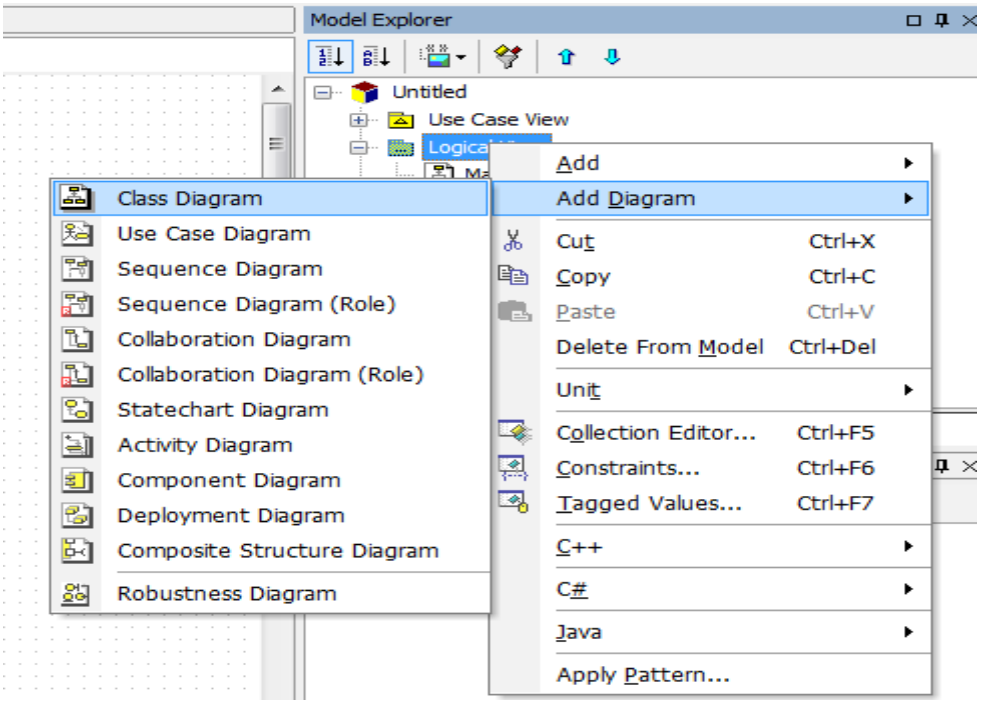


Рисунок 38. Добавление диаграммы классов

Будет создана новая диаграмма классов со стандартным именем ClassDiagram1, которое можно изменить в редакторе свойств диаграммы (рис. 39).

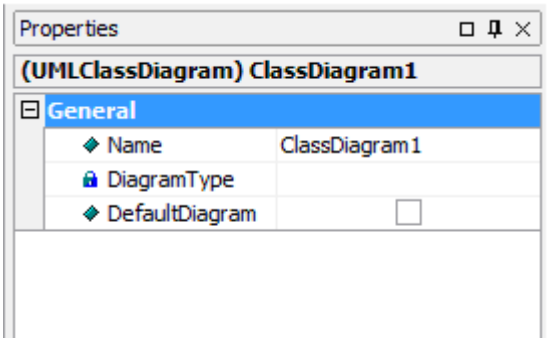


Рисунок 39. Редактор свойств диаграммы классов: изменение имени диаграммы

Пример. Рассмотрим сценарий Оформление заказа, который является внутренним потоком для прецедента Заказ товара. Опишем его

классы.

Данный сценарий позволяет покупателю оформить заказ из корзины и оплатить его с использованием банковской карты.

Давайте представим, как выполняется этот сценарий, еще раз. Покупатель, находясь в своей покупательской корзине, приняв решение о том, что он готов сделать заказ в магазине «Style», выбирает опцию «Оформить заказ». Как реагирует система на действия покупателя? Запускается сценарий Оформление заказа. Пользователь должен на специальной форме внести свои личные данные, подтвердить заказ или нет, и в зависимости от этого произвести оплату, затем получить подтверждение заказа. В системе появляется новый объект – заказ покупателя.

Выбор граничных классов.

По всей видимости, нам нужен будет хотя бы один граничный класс, который осуществляет связь между действующим лицом Покупатель и дополнительным прецедентом Оформить заказ. Назовем его ОформлениеЗаказа (PlaceOrder). Этот класс знает, какие товары и в каком количестве были в корзине покупателя, их нужно перенести в заказ. Также этот класс может знать, пуста корзина покупателя или нет, и, если пуста, то вывести об этом соответствующее сообщение. Для того, чтобы сделать заказ, покупатель должен ввести свои личные данные, электронный адрес, телефон и данные кредитной карты. Для этих целей мы введем еще один класс ВводЛичныхДанных (EnterPersonalInformation). После того, как выбраны товары и введена личная информация покупателя, остается только проверить детали заказа и согласиться с ними или нет – для этого действия введем класс ПроверкаДеталейЗаказа (ConfirmOrder). Наконец, когда покупатель завершит оформление заказа, то на экране он видит номер заказа и подтверждение заказа отправляется покупателю на электронный адрес, для выполнения этих обязанностей создадим еще один граничный класс ПодтверждениеЗаказа (OrderConfirmation)

Выбор управляющих классов.

Создадим один управляющий класс, который будет распределять обязанности других классов и вызывать их операции при выполнении данного сценария. Назовем этот управляющий класс МенеджерОформленияЗаказа (PlaceOrderManager).

Выбор классов-сущностей.

В сценарии Оформление заказа речь идет о покупателе, заказе и товарах. Создадим классы-сущности Покупатель (Customer), Заказ (Order), Товар (Item).

Возможно, что в ходе дальнейшего проектирования какие-то новые

классы будут добавлены для этого сценария, а какие-то, напротив, из этой диаграммы удалены.

Построим диаграмму классов сценария Оформление заказа в StarUML.

Создайте в пакете Logical View диаграмму классов описанным выше способом. Переименуйте ClassDiagram1 в Place Order (Оформление Заказа).

Создадим новый класс Покупатель (Customer). Щелкните правой кнопкой мыши по Logical View в навигаторе модели, в контекстном меню выберите пункт Add (Добавить), затем выберите пункт Class (Класс). Новый класс будет создан и отобразится в навигаторе модели. На вкладке Properties (Свойства) измените имя класса на Покупатель (Customer). *Заметим, что при таком способе создания класса, StarUML создает новое пространство имен.* Если вы хотите создать класс, который носит такое же имя, как, например, действующее лицо на диаграмме прецедентов, то для того, чтобы StarUML «разрешил» использовать это имя еще раз, нужно создать класс способом, описанным выше. Теперь перетащите этот класс из навигатора модели на лист диаграммы Place Order (рис. 40).

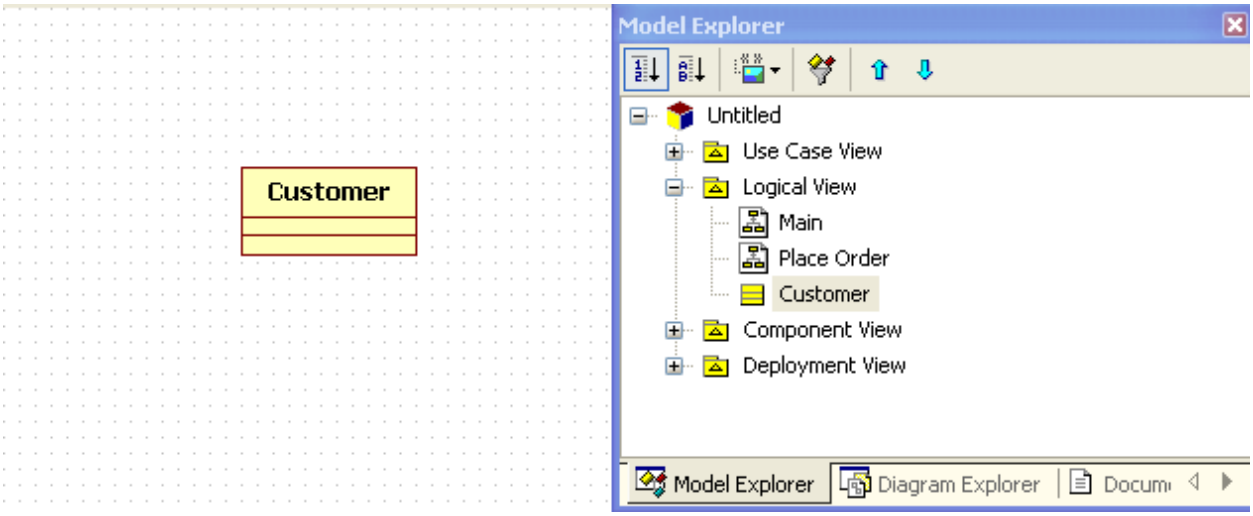


Рисунок 40. Создание класса Покупатель

Для того чтобы создать класс ОформлениеЗаказа (PlaceOrder), используем другой метод. Слева на панели инструментов (Toolbox) щелкните изображение класса (Class), затем щелкните на листе диаграммы Place Order в том месте, куда необходимо поместить класс, при этом стандартное имя класса станет активным, давая возможность его изменить. Измените имя класса на ОформлениеЗаказа (PlaceOrder) (рис. 41).

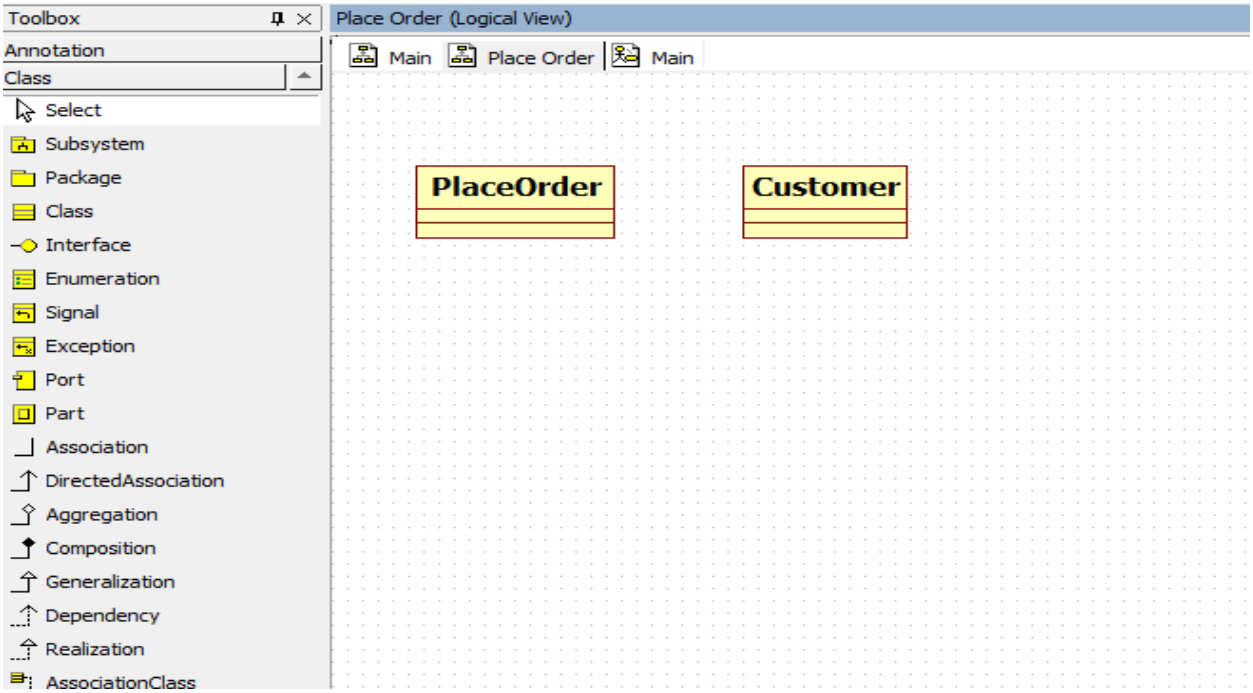


Рисунок 41. Создание класса ОформлениеЗаказа

Создадим все остальные классы, наша диаграмма классов представлена на рисунке 42.

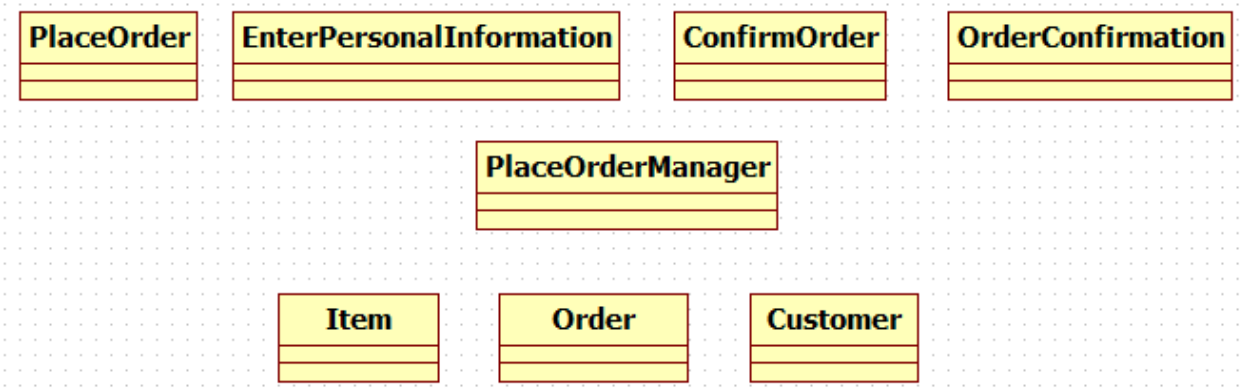


Рисунок 42. Диаграмма классов сценария Оформление заказа

Замечание. Мы выполняем диаграммы классов и взаимодействия на английском языке, так как эти диаграммы участвуют в генерации программного кода системы. Конечно, если мы бы мы выполнили их на русском языке, ничего страшного не случилось бы, но сгенерированный код содержал бы имена на кириллице, и для дальнейшего использования программного кода нам пришлось бы их все заменить. Для удобства читателя мы приводим соответствующие переводы фраз и названий.

6.5 Назначение стереотипов

Чтобы присвоить классу один из стереотипов UML, нужно выделить класс щелчком правой кнопки мыши, открыть редактор свойств Properties на

инспекторе модели и выбрать раздел стереотип Stereotype (рис. 43).

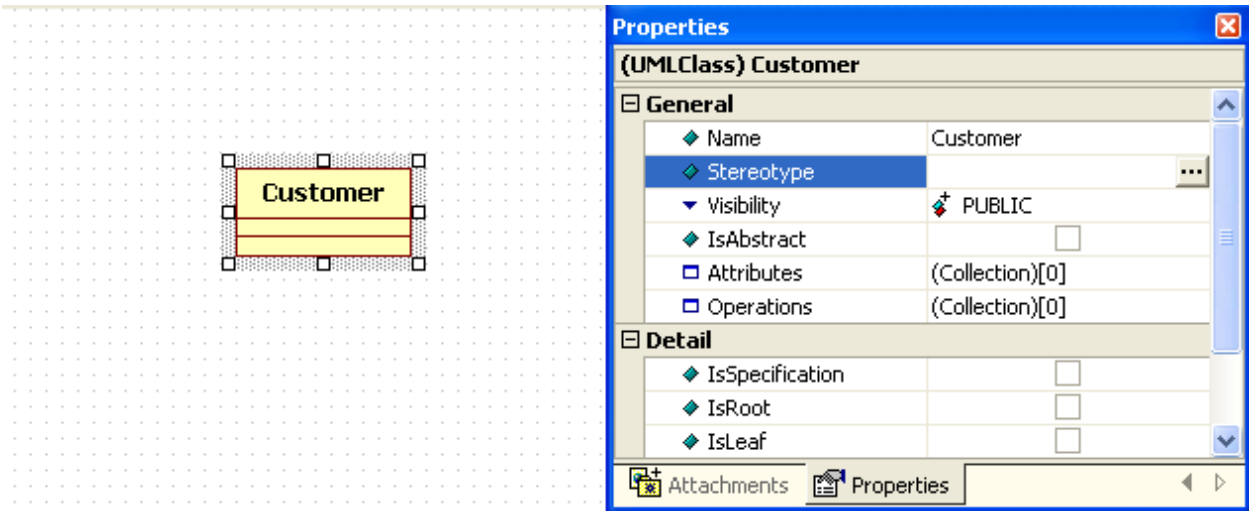


Рисунок 43. Назначение стереотипа классу

Если мы нажмем на значок ..., то в появившемся диалоге нам будет представлен список доступных стереотипов с кратким описанием и графическим обозначением. Для того чтобы присвоить классу стереотип, нужно выбрать его в списке, выделить и нажать кнопку OK (рис. 44).

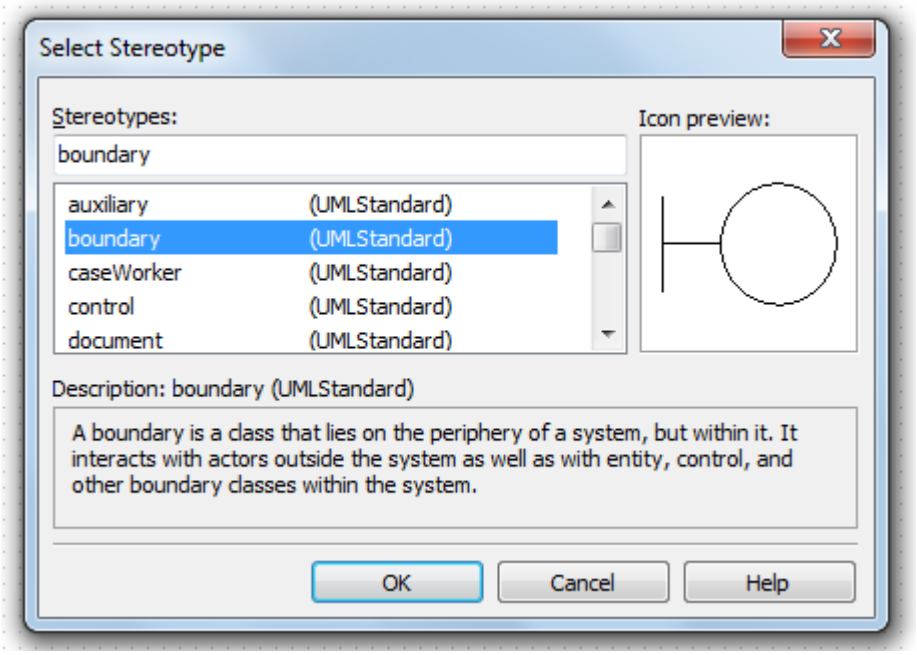


Рисунок 44. Выбор стереотипа

После присвоения классу стереотипа его внешний вид изменится. Рядом с именем класса появится имя стереотипа, заключенное в угловые скобки (рис. 45).

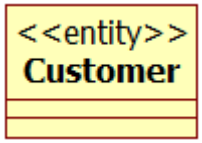


Рисунок 45. Отображение стереотипа класса

Пример. Присвоим классам сценария Оформление заказа соответствующие стереотипы. Диаграмма классов изменится (см. рис. 46).

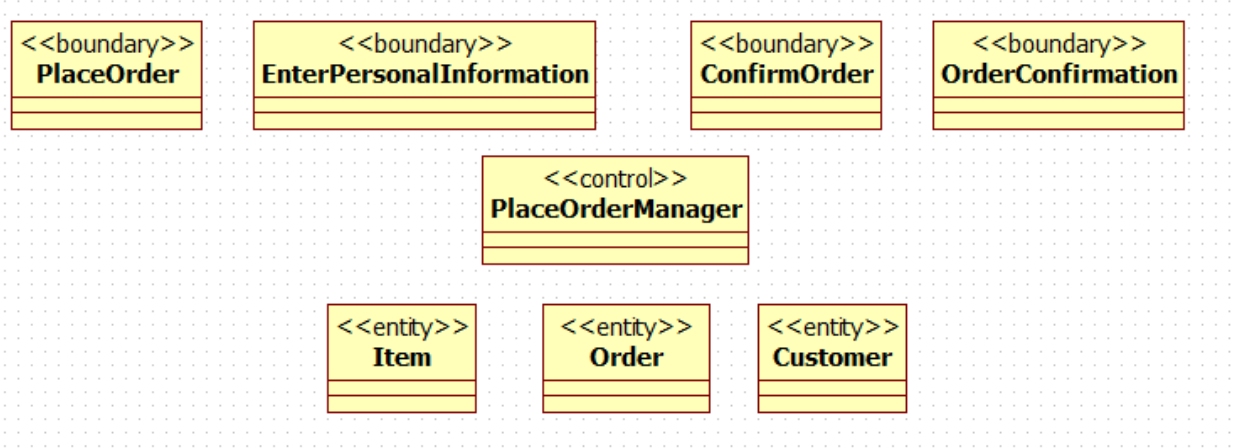


Рисунок 46. Диаграмма классов сценария Оформление заказа со стереотипами

Мы можем отобразить стереотипы классов с помощью пиктограмм. Для этого нужно выделить класс, щелкнуть по выделенной области правой кнопкой мыши, в контекстном меню выберите пункт Format, затем выберите пункт Stereotype Display, далее в списке выберите Iconic (рис.47).

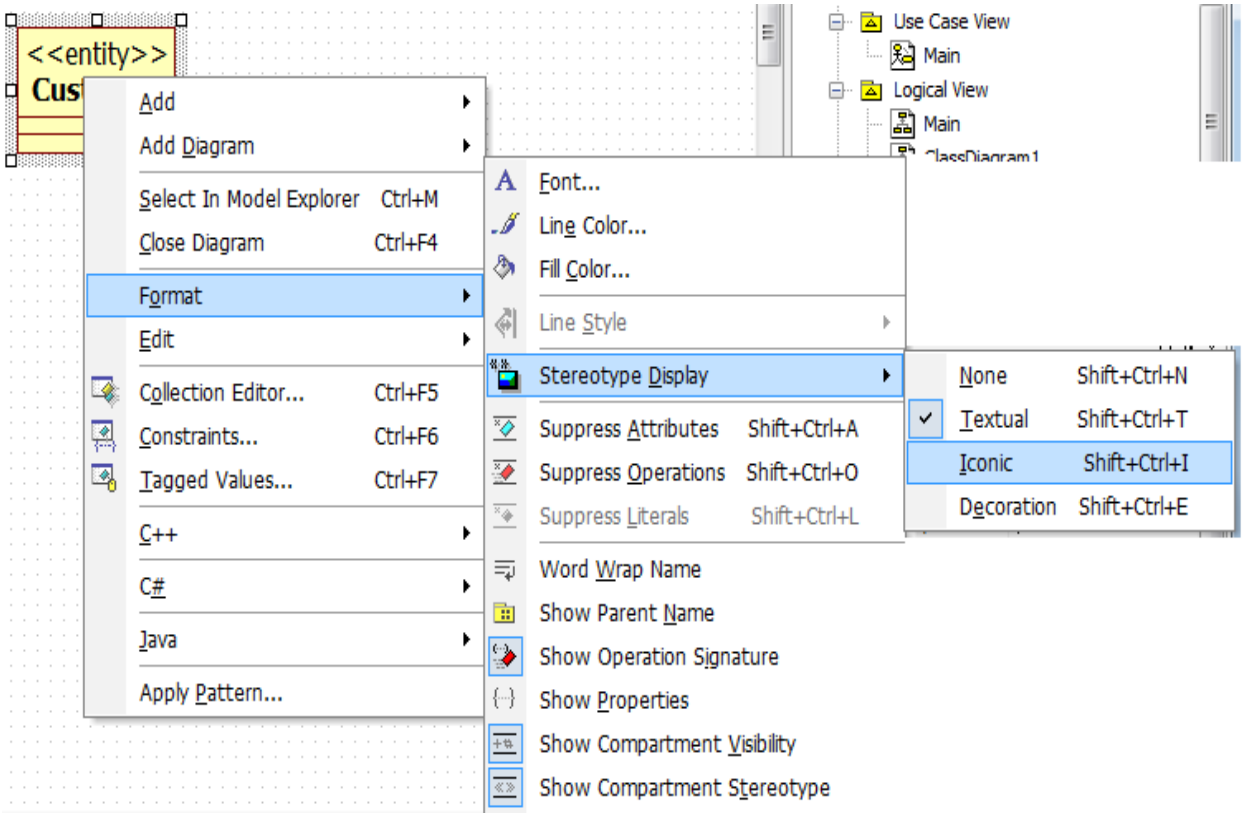


Рисунок 47. Задание отображения стереотипов классов в виде пиктограмм

Классы будут отображаться как пиктограммы (рис. 48).

Рисунок 48. Отображение классов с помощью пиктограмм стереотипов

Пример. Диаграмму классов сценария Оформление заказа отобразим с помощью пиктограмм (см. рис. 49).

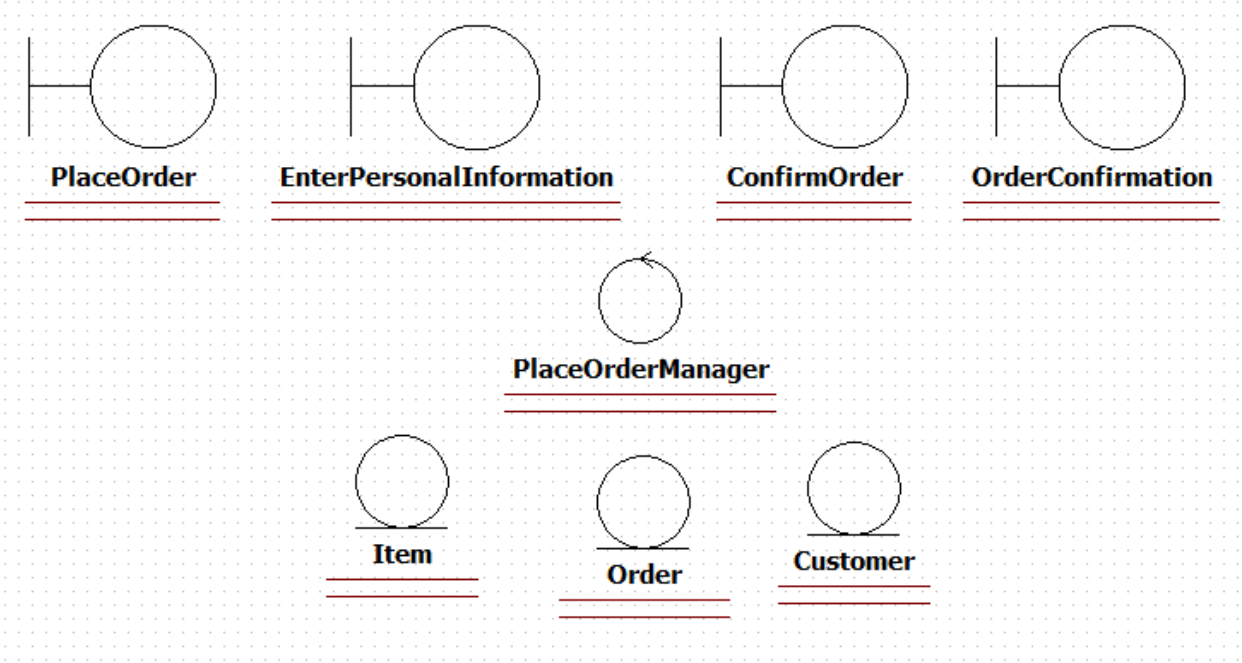


Рисунок 49. Диаграмма классов сценария Оформление заказа в пиктограммах

7. Пакеты в языке UML

Если в нашей модели немного классов, то нам легко ими управлять, однако многие системы содержат большое количество классов, поэтому необходим механизм, позволяющий классы группировать и облегчающий их повторное использование. Таким механизмом в UML являются пакеты.

Пакет (package) — общецелевой механизм для организации различных элементов модели в группы.

Подпакет (subpackage) — пакет, который является составной частью другого пакета.

Пакет в логическом представлении модели – это объединение классов или других пакетов. С помощью объединения классов в пакеты мы можем получить представление о системе на более высоком уровне. Напротив, рассматривая пакет, мы получаем более детальное представление модели.

Объединять классы в пакеты можно как угодно, однако, существует несколько наиболее распространенных подходов.

1. можно группировать классы по стереотипам: классы-сущности, граничные и управляющие классы.
2. группировка классов по их функциональности: например, пакет классов, отвечающих за безопасность системы или пакет классов Работа с сотрудниками и т.п.
3. наконец, применяют комбинацию двух указанных методов.

В дальнейшем можно вкладывать пакеты друг в друга.

Чаще всего пакет на диаграмме изображается в виде папки с закладкой с именем пакета (рис. 50).

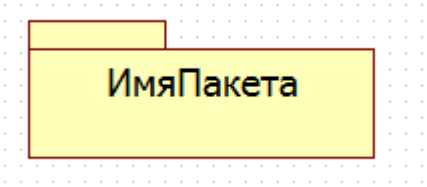


Рисунок 50. Пакет

Главную диаграмму пакетов системы обычно помещают диаграмме Main представления Logical View. Для того чтобы создать пакет на диаграмме, нужно открыть рабочее поле диаграммы, щелкнуть по элементу пакет Package на панели элементов слева, затем щелкнуть по рабочему полю диаграммы в том месте, где вы хотите поместить пакет. В окне редактора свойств можно задать новое имя пакета.

Чтобы разместить классы по пакетам, используют метод перетаскивания: на навигаторе модели нужно перетащить, удерживая левую кнопку мыши, классы в соответствующие пакеты на навигаторе модели.

Пример. Созданные ранее классы сценария Оформление заказа сгруппируем по пакетам, диаграмму пакетов поместим на листе Main представления Logical View. Создадим пакеты Граничные классы, Классы-сущности, Управляющие классы (рис. 51).

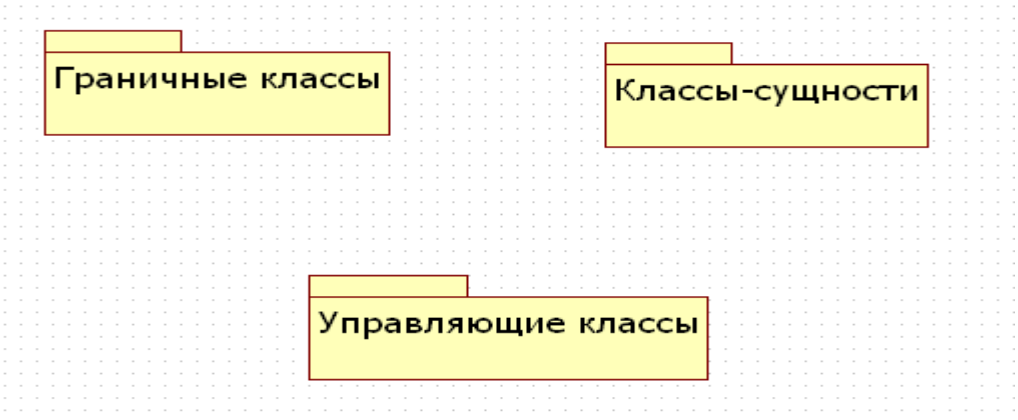


Рисунок 51. Диаграмма пакетов

Перетащим на навигаторе модели класс Customer (Покупатель) в пакет Классы-сущности (рис. 52).

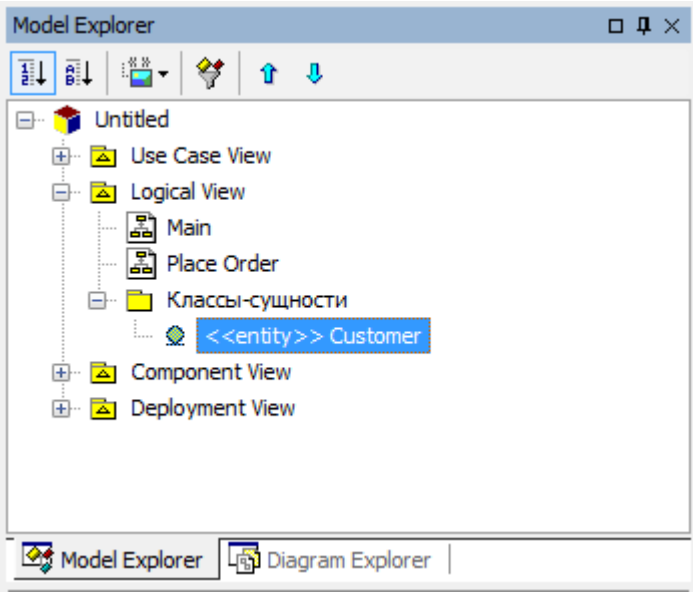


Рисунок 52. Перемещение классов в пакеты

Аналогично классы Item (Товар) и Order (Заказ) разместим в пакете Классы-сущности, классы ОформлениеЗаказа (PlaceOrder), ВводЛичныхДанных (EnterPersonalInformation), ПроверкаДеталейЗаказа (ConfirmOrder) и ПодтверждениеЗаказа (OrderConfirmation) разместим в пакете Граничные классы, а класс МенеджерОформленияЗаказа (PlaceOrderManager) – в пакете Управляющие классы.

Обратите внимание, что при этом внешний вид диаграмм классов Place Order и диаграммы пакетов Main не изменится.

8. Диаграммы взаимодействия

Диаграммы взаимодействия отображают один из процессов обработки информации в варианте использования: какие объекты нужны потоку, какими сообщениями обмениваются объекты, какие действующие лица инициируют поток и в какой последовательности отправляются сообщения. Для одного потока событий может быть построено несколько диаграмм взаимодействия.

Основной элемент диаграмм взаимодействия – это объект.

Объектом описывают нечто содержащее в себе данные и поведение. Это термин, описывающий реальные, конкретные предметы или абстрактные сущности.

Объекты изображаются в виде прямоугольников, имена объектов

подчеркиваются. Внутри прямоугольника, обозначающего объект, записывается с большой буквы имя объекта. Имя объекта подчеркивается. Если оно содержит несколько слов, то все они должны начинаться с большой буквы.

Пример. Поскольку наша система предназначена для заказа товаров, то скорее всего нам придется построить диаграмму взаимодействия с участием такого объекта, как Товар (например, диаграмму, описывающую добавление товара в корзину). Мы можем использовать на диаграмме взаимодействия какой-то конкретный объект-товар, например, Галстук-Бабочка. А можем назвать этот объект безлично – Товар (см. рис. 53).

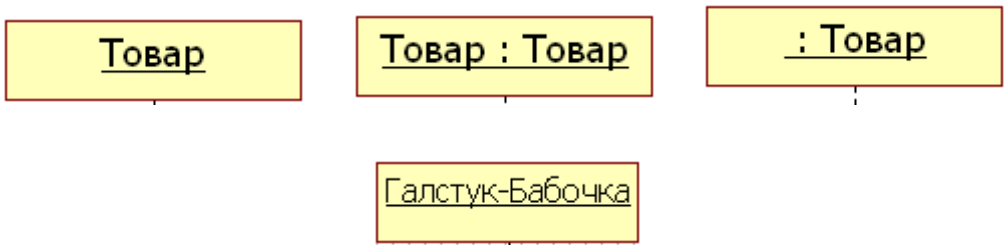


Рисунок 53. Примеры именования объектов

Объекты, помещаемые на диаграммы взаимодействия, скорее всего будут объектами системы и будут относиться к программному обеспечению. При проектировании этих диаграмм можно представлять себе объекты, как экранные формы или части приложений, отвечающие за выполнение определенных действий, или объектом может быть запись в таблице базы данных.

Если перед тем, как строить диаграммы взаимодействия мы построили диаграммы классов, то тогда поиск объектов упрощается. Объекты соответствуют своим классам или их операциям, и мы можем создавать и располагать их на диаграмме последовательности действий или кооперативной диаграмме.

Если мы хотим изучить взаимодействие объектов до того, как переходить к поиску классов, то поиск объектов можно начать с изучения имен существительных в потоке событий. Многие из них станут хорошими кандидатами в объекты. Мы также можем выделить объекты-сущности, граничные объекты и управляющие объекты на основе выбранных классов.

Существует **два типа** диаграмм взаимодействия – диаграммы последовательности (или последовательности действий) и диаграммы кооперации. Первые отображают обмен сообщениями между объектами во времени, а вторые отображают структуру взаимодействия. На обеих диаграммах отображается одна и та же информация, но разными способами:

диаграмма последовательностей изображает поток управления, а кооперативная диаграмма – поток данных.

8.1 Диаграммы последовательности

Как правило, поток событий описывает не одну последовательность действий, а несколько возможных, это отражается наличием главного потока событий и альтернативных потоков. Чаще всего невозможно описать прецедент с помощью только одной последовательности действий. Например, для прецедента Заказ товаров возможно оформление заказа без изменения корзины, с изменением состава корзины, или покупатель, просмотрев корзину, захочет вернуться в каталог и что-то в нее добавить, возможно, вернувшись в каталог, покупатель не станет ничего больше добавлять, а снова вернется в корзину и оформит заказ. Каждый такой вариант мы можем описать своей последовательностью действий, своим сценарием. И, таким образом, один прецедент описывает несколько последовательностей – сценариев, каждый из которых описывает один из вариантов возможного потока событий.

Сценарий (Scenario) – это некоторая последовательность действий, иллюстрирующая поведение системы [2].

Сценарий – это экземпляр потока событий. Он представляет собой одиночный проход по потоку событий для прецедента. Для графического отображения сценария используются диаграммы последовательностей.

Диаграмма последовательности действий отображает взаимодействие объектов, упорядоченное по времени.

8.1.1 Основные элементы нотации диаграмм последовательности

На диаграммах последовательности изображаются объекты, классы и последовательность сообщений, которыми обмениваются объекты в ходе выполнения сценария (рис. 54).

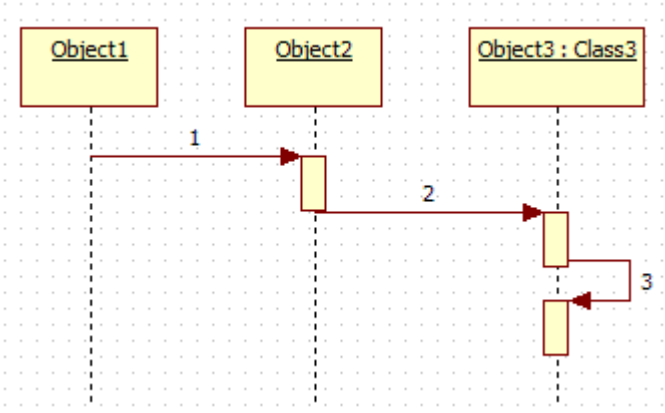


Рисунок 54. Общий вид диаграммы последовательности

На диаграмме последовательностей могут также изображаться экземпляры действующих лиц. Для того чтобы поместить действующее лицо на диаграмму, нужно найти его в навигаторе модели справа и перетащить на поле диаграммы последовательностей (рис. 55).

Пример.

Рисунок 55. Экземпляр действующего лица на диаграмме последовательности

Действующие лица, присутствующие на диаграммах взаимодействия, выделяются из потока событий как сущности, запускающие процессы. На одной диаграмме их может быть несколько.

Для того чтобы поместить экземпляр уже созданного ранее на диаграмме прецедентов действующего лица на диаграмму взаимодействия, просто перетащите его с навигатора модели на рабочее поле диаграммы.

Каждый объект или действующее лицо на диаграмме последовательностей имеет свою линию жизни, которая обозначается пунктиром.

Линия жизни объекта (object lifeline) – вертикальная пунктирная линия на диаграмме последовательности, которая представляет существование объекта в течение определенного периода времени.

Фокус управления (активность, focus of control) - специальный символ на диаграмме последовательности, указывающий период времени, в течение которого объект выполняет некоторое действие, находясь в активном состоянии.

Фокус управления изображается тонким прямоугольником, расположенным на линии жизни (рис. 56).

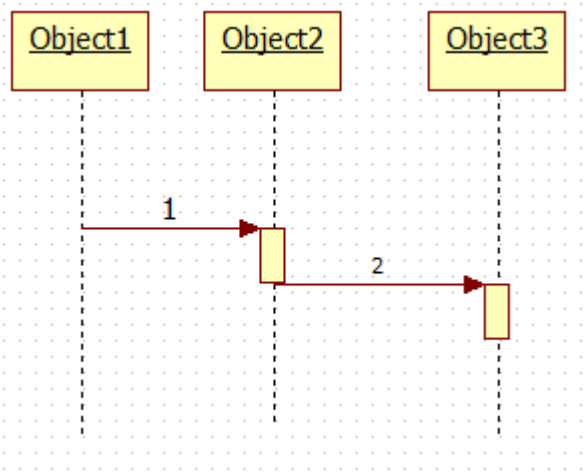


Рисунок 56. Фокус управления

Иногда отображение фокуса активности и нумерации сообщений на диаграмме могут сделать ее трудной для чтения. Чтобы фокус управления и нумерация сообщений не отображались на диаграмме последовательности в StarUML нужно открыть редактор свойств этой диаграммы в инспекторе модели и в разделах ShowSequenceNumber и ShowActivation убрать «галочки» (рис. 57).

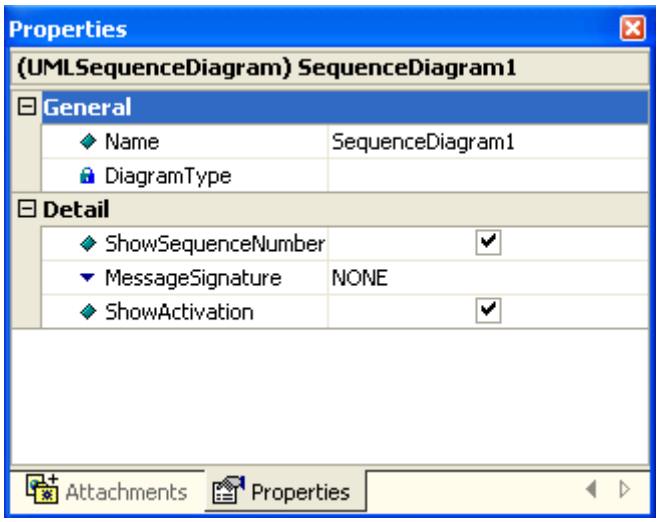


Рисунок 57. Управление отображением фокуса управления и нумерации сообщений

Объекты и действующие лица на диаграммах последовательности обмениваются сообщениями. Сообщения обозначаются стрелками, идущими от отправителя к получателю.

Сообщение (***message***) — спецификация передачи информации от одного элемента модели к другому с ожиданием выполнения определенных действий со стороны принимающего элемента (рис. 58).

Пример.

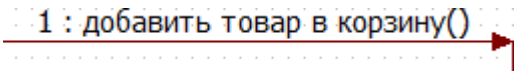


Рисунок 58. Сообщение

Для сообщений на диаграммах последовательностей, как и для других элементов модели, доступен ряд спецификаций.

Во-первых, у каждого сообщения должно быть имя, соответствующее его цели.

Во-вторых, сообщения на диаграммах последовательностей можно соотнести с операциями, определенными для классов. Если от одного объекта к другому направлено сообщение, то это означает, что объект-источник вызывает операцию объекта-приемника. Объект не может вызвать произвольную операцию: она должна быть доступна этому объекту.

В особых случаях сообщение не становится операцией: например, ввод логина и пароля подразумевает их печать в соответствующих полях, и сообщение будет реализовано в виде поля ввода в окне программы [1].

Процедура создания операций из сообщений будет описана ниже.

В-третьих, мы можем для каждого сообщения установить тип синхронизации. Каждому типу соответствует его обозначение.

Вызов операции (процедуры) (call) вызывает операцию того объекта, к которому направлено. Объект может вызвать свою операцию. Тогда стрелка начинается и заканчивается на линии жизни одного и того же объекта, такое сообщение называется ***рефлексивным***.

Синхронное сообщение обозначается стрелкой с закрашенным наконечником (рис. 59).

Рисунок 59. Синхронное сообщение

Асинхронное сообщение (send) посылает объекту сигнал. При этом источник не ждет отклика приемника или подтверждения получения, а продолжает свою работу. Обозначается нежирной стрелкой (рис. 60).

Рисунок 60. Асинхронное сообщение

Ответное сообщение (return) возвращает значение из процедуры тому объекту, к которому направлено. Обозначается пунктирной стрелкой (рис. 61)

Рисунок 61. Ответное сообщение

Создать объект (create) – создает новый объект. Обозначается стрелкой со стереотипом <<create>> (рис. 62).

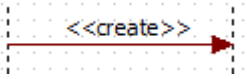


Рисунок 62. Создание объекта

Уничтожить объект (destroy)- удаляет объект. Объект может уничтожить сам себя. Обозначается стрелкой со стереотипом <<destroy>>. При уничтожении объекта на его линии жизни появляется символ разрушения, который обозначается крестом (рис. 63).

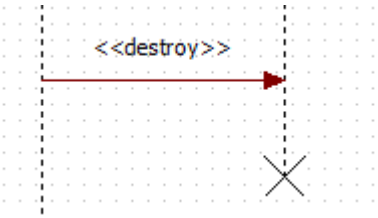


Рисунок 63. Уничтожение объекта

Для определения типа сообщения в StarUML нужно выполнить след действия: выделите сообщение, щелкнув по соответствующей стрелке один раз левой кнопкой мыши, откройте редактор свойств, выберите на нем раздел ActionKind и в выпадающем списке выберите тот тип синхронизации, который вы хотите установить (рис. 64).

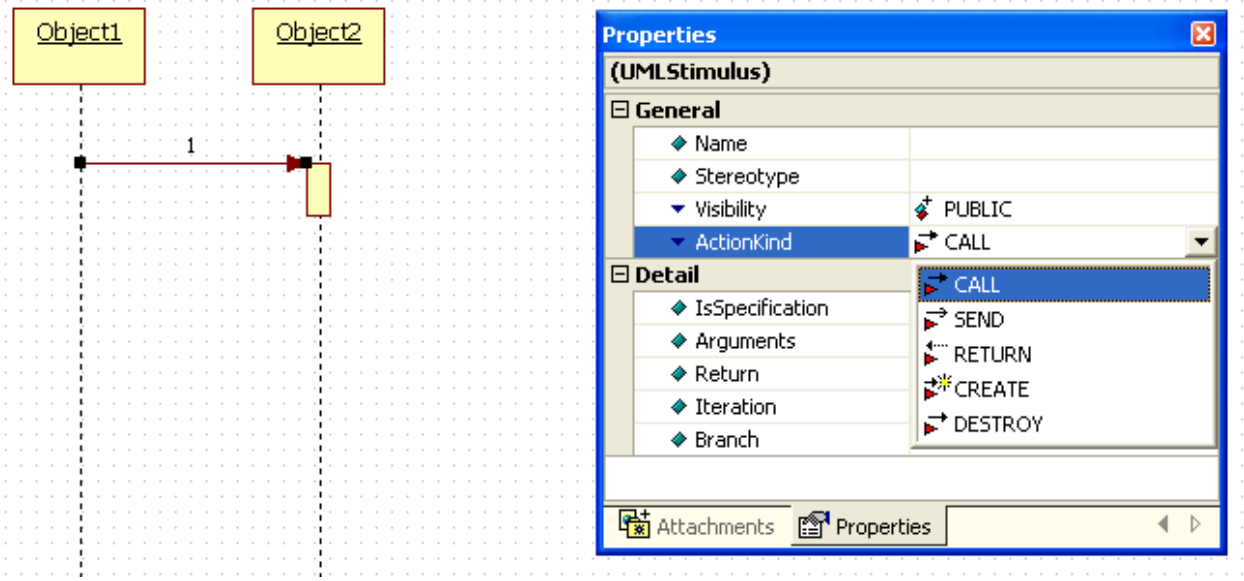


Рисунок 64. Выбор типа сообщения

8.1.2 Добавление диаграммы последовательности в модель

Для создания новой диаграммы последовательности нужно выполнить следующие шаги: щелкнуть правой кнопкой мыши по папке представления Logical View в навигаторе модели, в контекстном меню выбрать пункт Add Diagram, в списке выбрать диаграмму последовательности Sequence Diagram (рис. 65).

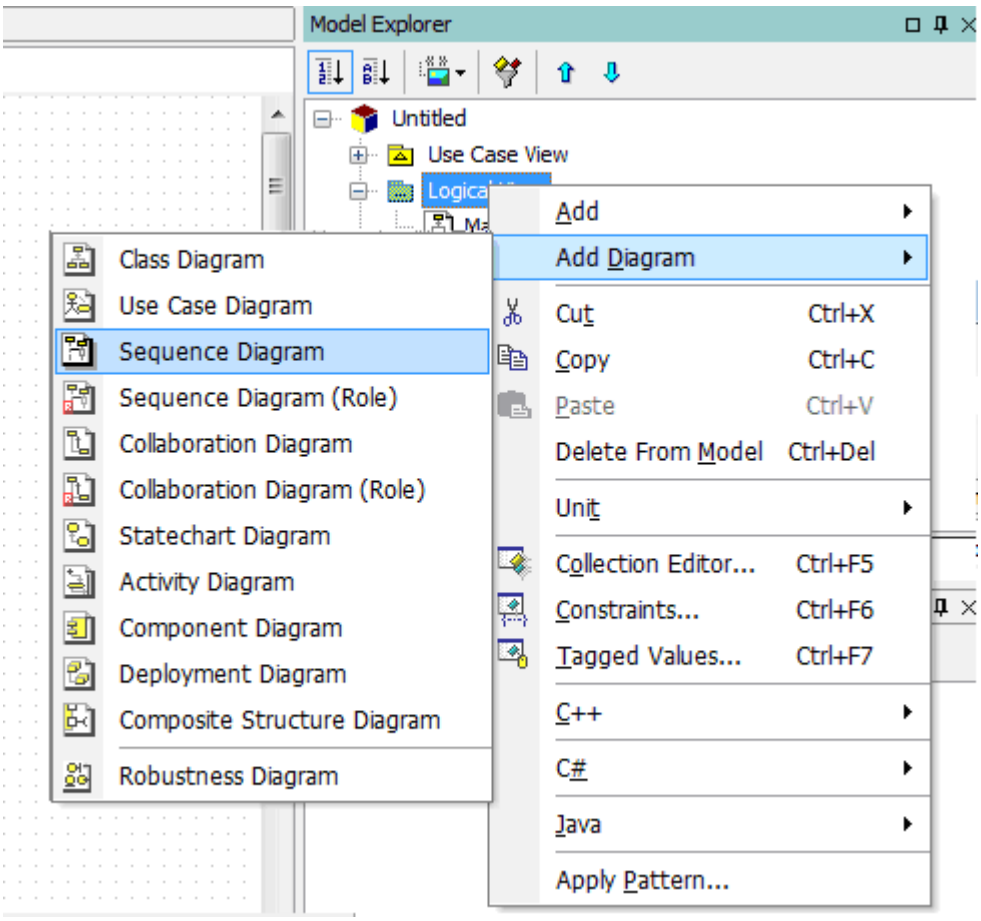


Рисунок 65. Добавление диаграммы последовательности

Мы также можем использовать диаграмму последовательности для детализации прецедента. Для этого нужно связать диаграмму с прецедентом: для создания диаграммы щелкните правой кнопкой мыши по прецеденту, а не по папке Logical View. Однако, если мы строим диаграмму последовательности для анализа системы, то лучше все-таки помещать ее в Logical View.

Пример. Мы уже определили классы сценария Оформление заказа, теперь с помощью диаграммы последовательности покажем, как взаимодействуют объекты этих классов во времени.

Составим диаграмму последовательности для случая, когда покупатель успешно оформляет заказ (рис. 66).

Покупатель выбирает опцию «Оформить заказ» (place order), при этом вызывается некоторый объект PlaceOrder (забегая вперед скажем, что это будет граничный объект, принадлежащий соответствующему граничному классу). Далее открывается форма ввода личных данных покупателя и его кредитной карты (EnterPersonalInformation), на ней покупатель вводит свое имя, адрес, телефон, адрес электронной почты (enter personal information) и кредитные данные. Информация принимается и открывается форма

подтверждения заказа (ConfirmOrder), покупатель подтверждает, что согласен с реквизитами заказа (confirm order), детали заказа сохраняются для дальнейшего использования (save the details). Фокус управления передается некоторому управляющему объекту (PlaceOrderManager), который обращается к внешней кредитной системе (Credit System) для проведения платежа. Если платеж прошел успешно (а именно такой сценарий мы сейчас и рассматриваем), то PlaceOrderManager посылает сообщение (create order) создать объект Заказ (Order), затем вызывает форму подтверждения заказа (OrderConfirmation). Объект Заказ (Order) обращается к объектам Товар (Item) для того, чтобы получить информацию о товарах и создает заказ. Процесс завершается.

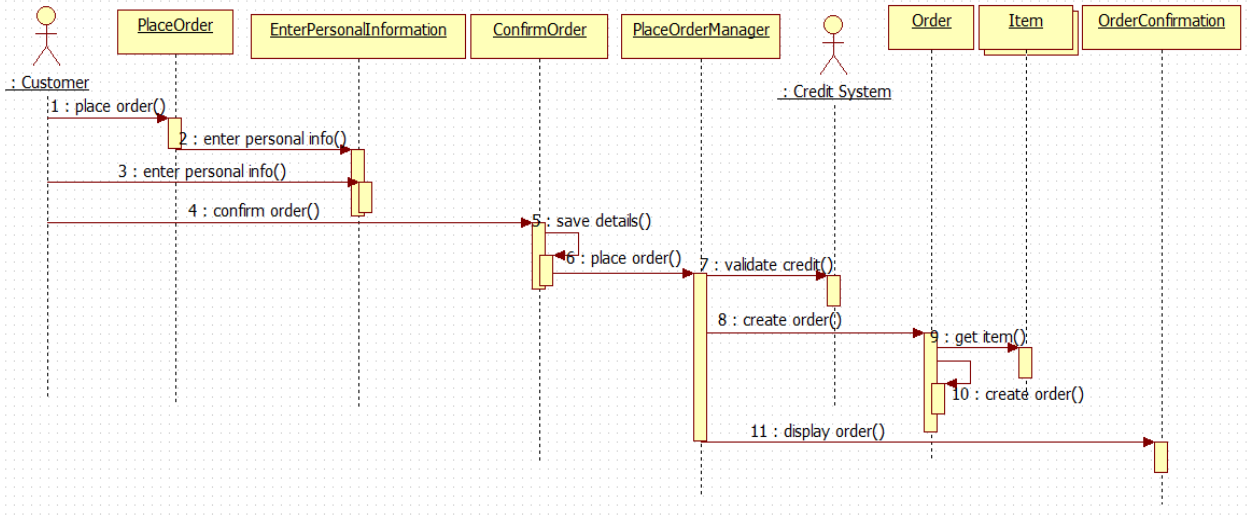


Рисунок 66. Диаграмма последовательности сценария Оформление заказа

Замечание. Обратите внимание, что символ объекта Товар (Item) на диаграмме последовательности отличается от символов других объектов. Дело в том, что мы задали множественный экземпляр класса. Действительно, заказ может состоять из нескольких товаров, значит объекту Заказ (Order) требуется получить информацию о нескольких объектах Товар (Item). Вместо того, чтобы представлять каждый товар отдельно мы используем нотацию UML для множественного экземпляра класса, представляя одним значком несколько объектов.

Чтобы сделать объект множественным в StarUML выделите объект, щелкнув по нему мышью один раз, в открывшемся редакторе свойств поставьте флажок в разделе IsMultiInstance (рис. 67).

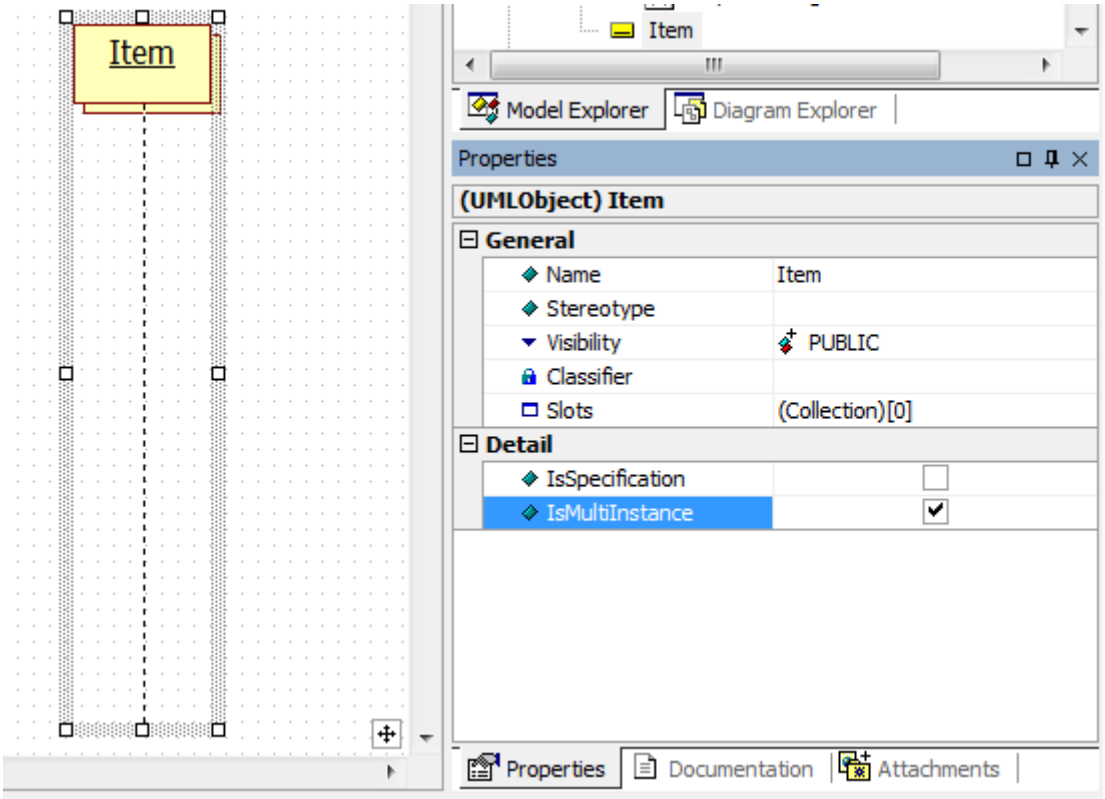


Рисунок 67. Создание множественного объекта

8.1.3 Ветвление потока управления

Обычно для основного потока событий большинства прецедентов строится одна диаграмма последовательности, для альтернативных потоков - дополнительные диаграммы, описывающие все остальные сценарии. Так поступают потому, что на диаграмме последовательности действий сложно отобразить логику ЕСЛИ-ТО-ИНАЧЕ. Однако если это необходимо и не загромождает диаграмму, то это можно сделать с помощью условий. Приведем пример.

Пример. В процессе оформления покупателем заказа возможны несколько альтернатив. Например, на втором шаге оформления заказа покупатель может подтвердить свой заказ, а может и не согласиться с его реквизитами (см. пример выше). На диаграмме последовательности это можно изобразить так, как это показано на рисунке 68.

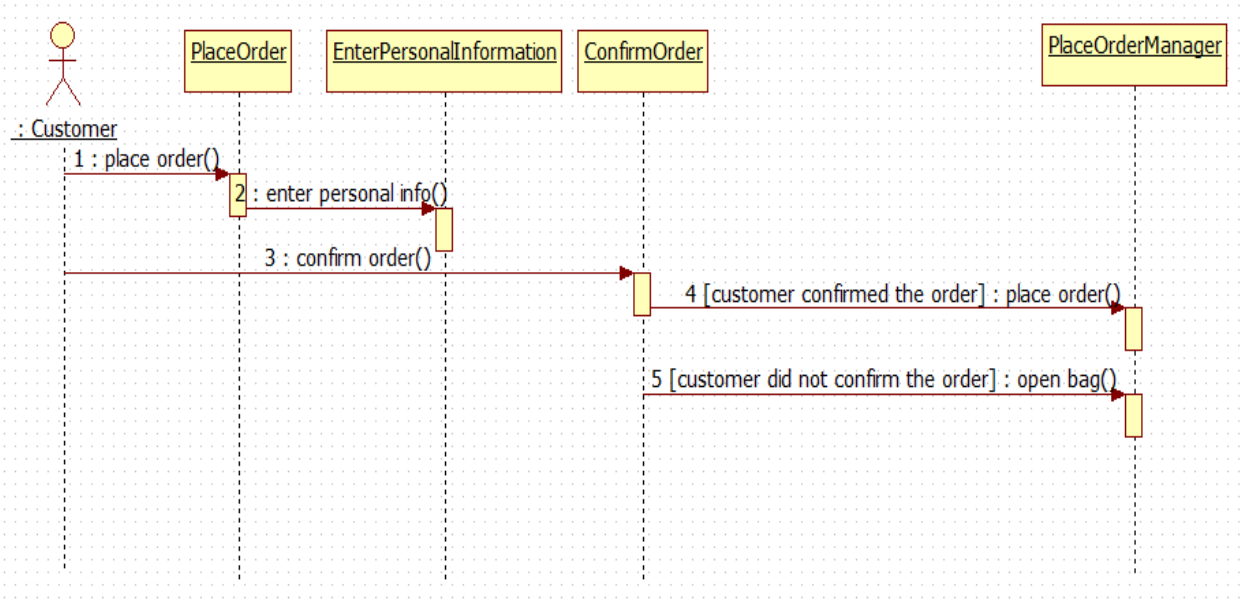


Рисунок 68. Ветвление потока управления

Если покупатель подтверждает свой заказ на втором шаге (customer confirmed the order), то процесс переходит оплате заказа. Если покупатель не подтверждает заказ (customer did not confirm the order), то открывается корзина покупателя. Условие, как это принято в нотации UML, записывается в квадратных скобках []. Обратите внимание, что мы упростили предыдущую диаграмму описания оформления заказа, иначе добавление ветвей процесса сделало бы ее громоздкой и трудно понимаемой. На практике лучше изображать диаграмму последовательности отдельно для каждого сценария потока событий.

8.2 Взаимосвязь диаграмм классов и последовательности

Процесс построения модели системы является итеративным. Особенно хорошо это можно видеть при создании диаграмм классов и последовательности. Какую диаграмму создавать первой: классов или последовательности? Одни разработчики начинают с диаграмм классов, другие – наоборот, с последовательности. И в том и в другом случае, скорее всего, обе эти диаграммы, построенные для одного сценария, будут в дальнейшем подвергаться изменению. После построения диаграмм последовательности на диаграммах классов могут появиться новые классы, а на диаграммах последовательности – новые объекты, которых раньше там не было, но они придут туда из диаграмм классов. Возможно, что некоторые объекты и классы будут, напротив, удалены.

Пример. В соответствии с нашей диаграммой последовательности на диаграмме классов сценария Оформить заказ произойдут некоторые

изменения.

Не сложно видеть на диаграмме последовательностей, что покупатель участвует в данном сценарии как действующее лицо-инициатор, запускающий выполнение сценария, но не как внутренний объект системы. Поэтому класс Customer (Покупатель) с данной диаграммы классов удалим: скорее всего такой класс в нашей модели будет (и мы удалили его только с диаграммы), но классом сценария Оформление заказа он не является.

Диаграмма классов прецедента Оформление заказа изменится и будет выглядеть так (рис. 69).

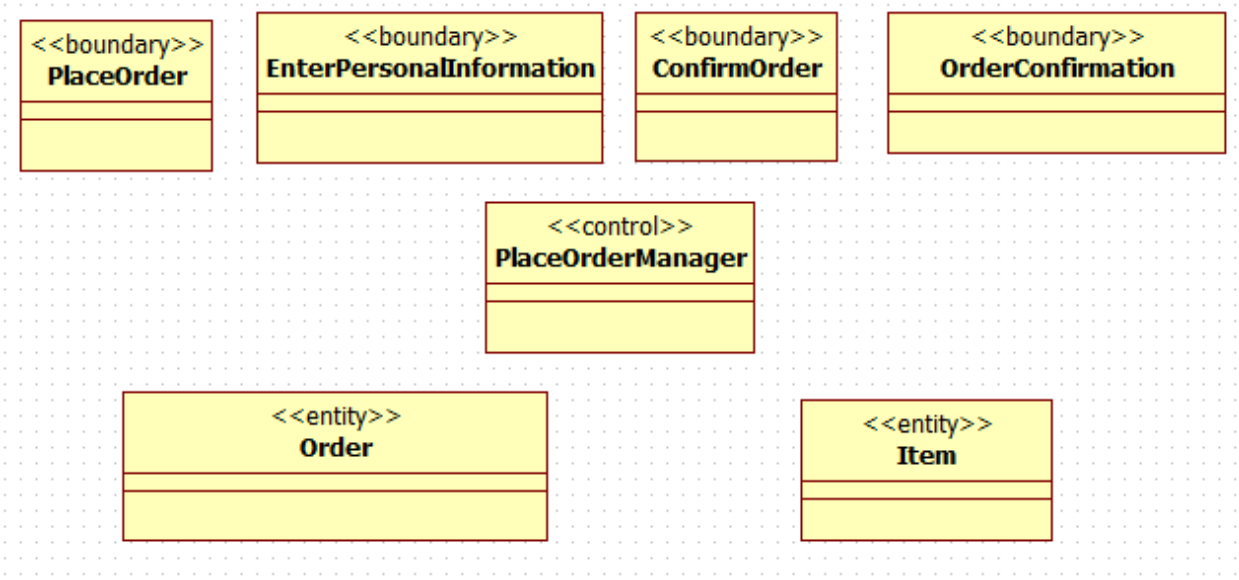


Рисунок 69. Измененная диаграмма классов сценария Оформление заказа

Замечание. Для создания диаграммы последовательностей, мы могли каждый объект этой диаграммы не создавать заново, а воспользоваться методом перетаскивания. Если перетащить класс с навигатора модели на диаграмму последовательности, то будет создан анонимный объект этого класса (рис. 70).

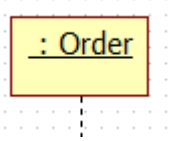


Рисунок 70. Анонимный объект класса Order

Можно изменить имя объекта, присвоив ему имя (рис. 71).

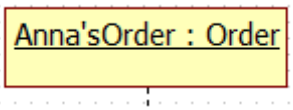


Рисунок 71. Именованный объект класса Order

8.3 Кооперативные диаграммы

Диаграмма кооперации – это альтернативный способ изображения сценария варианта использования. Этот тип диаграмм заостряет внимание на связях между объектами, отображая обмен данными в системе. А диаграммы последовательности отображают взаимодействие объектов во времени, поэтому ее следует читать сверху вниз и слева направо.

Диаграммы кооперации содержат все те же элементы, что и диаграммы последовательности: объекты, действующие лица, связи между ними и сообщения, которыми они обмениваются, но они уже не упорядочены во времени.

8.3.1 Добавление диаграммы кооперации в модель

Для того чтобы добавить диаграмму кооперации в представление Logical View, щелкните правой кнопкой мыши по папке содержащей диаграмму последовательности (если вы ее не переименовывали, то она носит имя CollaborationInstanceSet1), в контекстном меню выберите пункт Add Diagram, в списке выберите диаграмму кооперации Collaboration diagram (рис. 72).

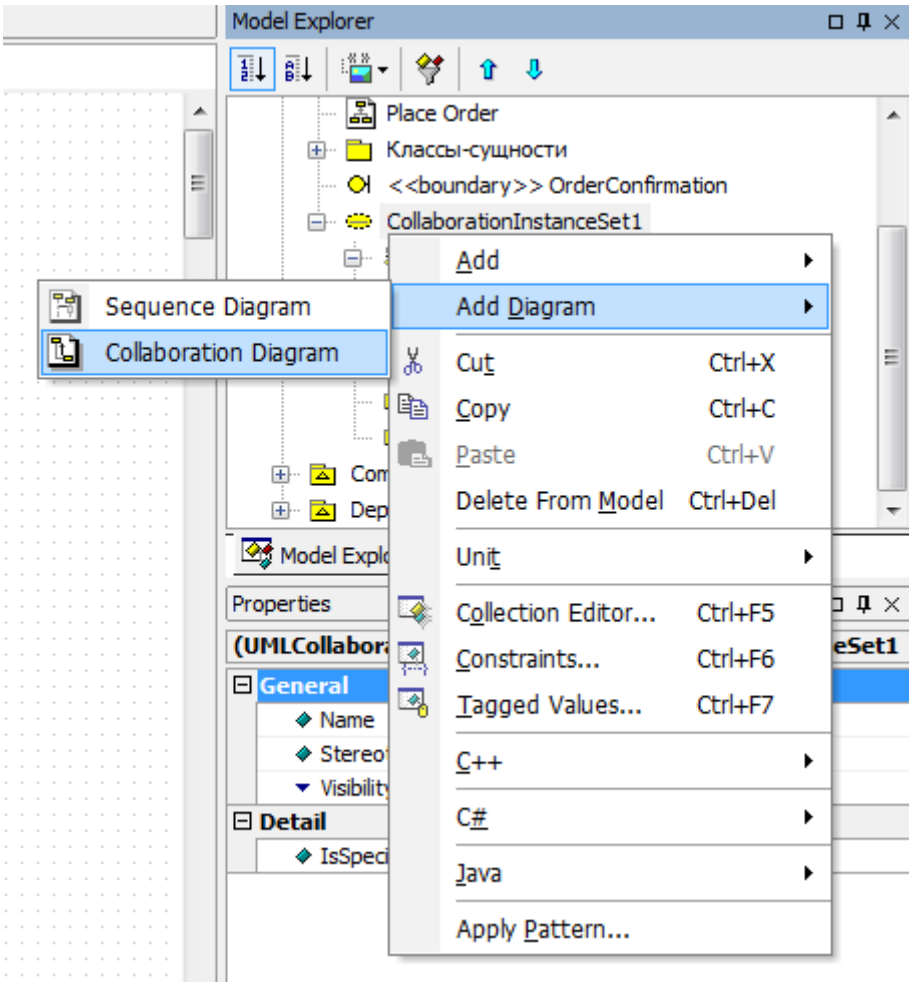


Рисунок 72. Добавление кооперативной диаграммы

Пример. Для сценария Оформление заказа, для которого мы уже составили диаграмму последовательности. На диаграмму кооперации поместим все те же объекты, перетащив их с навигатора модели (рис. 73).

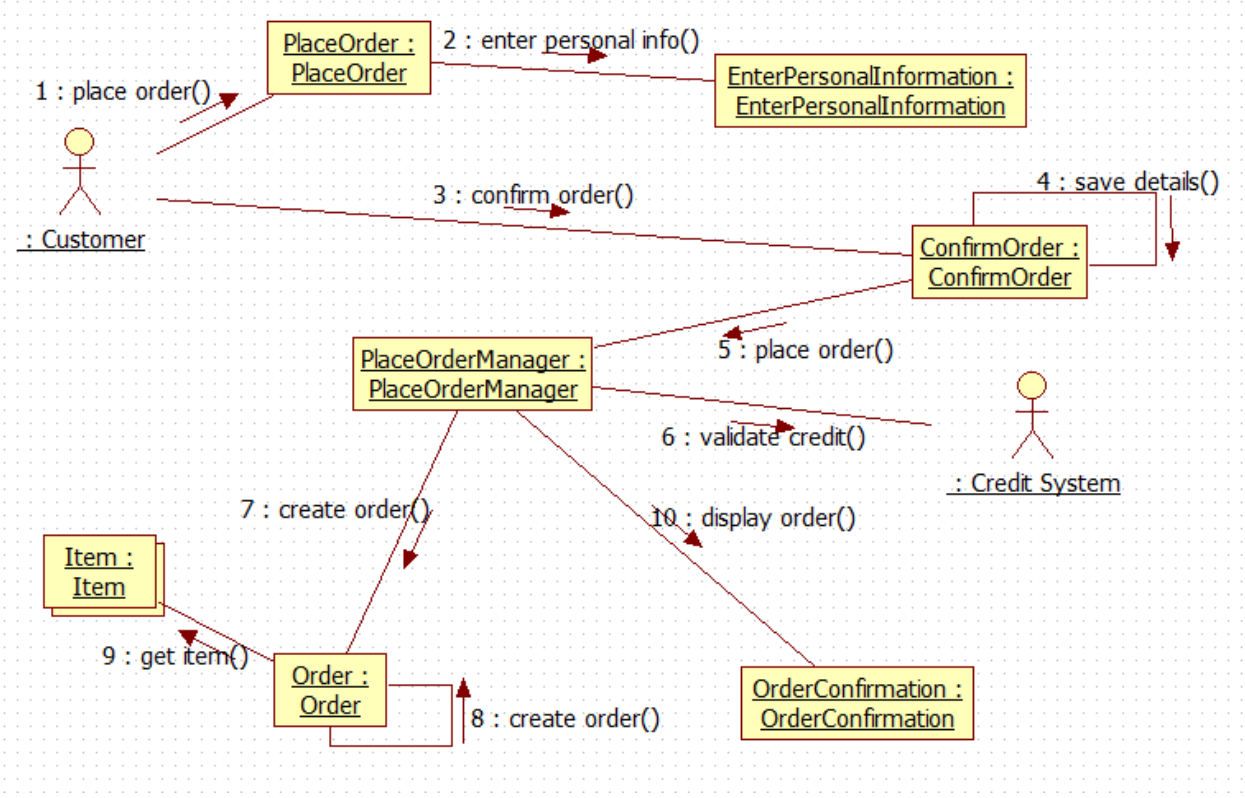


Рисунок 73. Кооперативная диаграмма сценария Оформление заказа

9. Атрибуты и операции классов

Механизм инкапсуляции в UML реализуется за счет объединения свойств и поведения в одном объекте. Свойства объекта описываются с помощью задания атрибутов класса, к которому относится объект, а поведение – заданием операций класса.

На прямоугольнике класса атрибуты описываются во второй секции под именем, а операции – в третьей, под атрибутами.

Атрибут класса служит для представления отдельного свойства или признака, который является общим для всех объектов данного класса. Атрибуты, таким образом, определяют структуру класса.

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

Пример. Класс Товар (Item) может содержать следующие атрибуты: артикул, название, цена, размерный ряд, цвет (рис. 74).

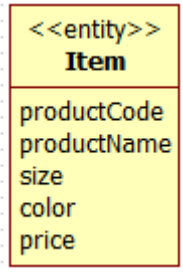


Рисунок 74. Класс Товар с атрибутами

9.1 Как создать атрибут класса в StarUML

Для того чтобы создать атрибут класса в StarUML, выделите класс, атрибуты которого вы хотите задать, щелкнув по нему один раз, затем в редакторе свойств Properties этого класса откройте раздел Attributes, нажав кнопку  . Откроется редактор коллекций (Collection Editor), содержащий вкладки редакторов атрибутов, операций и пр. (рис. 75).

Если в редакторе атрибутов нажать кнопку  , то будет создан новый атрибут и откроется редактор его свойств, в котором можно изменить имя созданного атрибута (раздел Name) и задать другие спецификации (о спецификациях атрибутов и операций будет сказано немного позже).

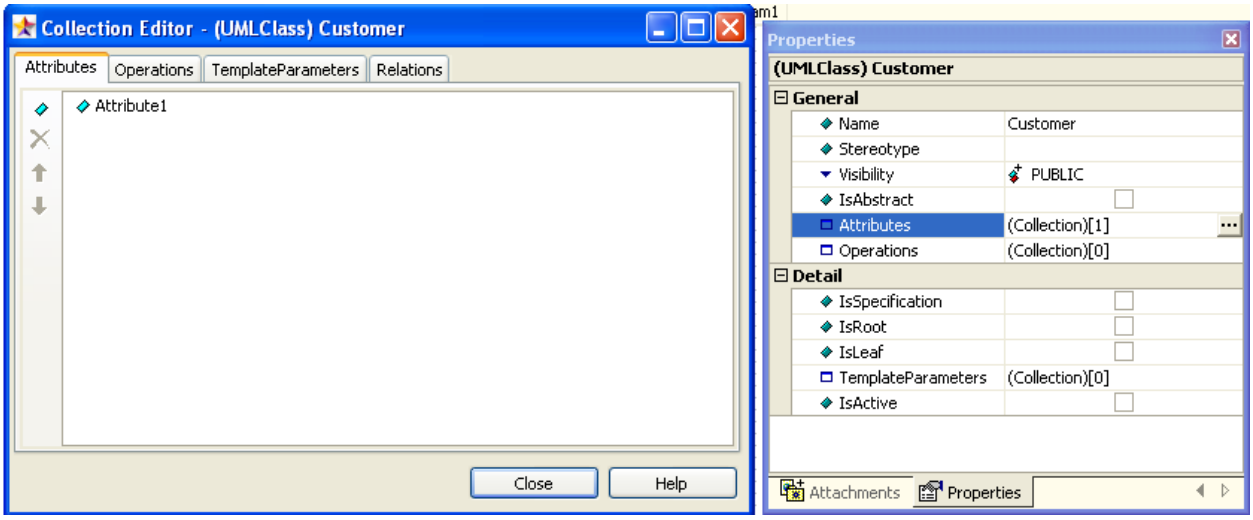
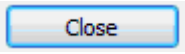


Рисунок 75. Создание атрибута

Для сохранения атрибута и его спецификаций просто закройте диалог, нажав кнопку 

.

Пример. Создадим атрибуты классов сценария Оформление заказа.

Для класса Заказ (Order) определим следующие атрибуты: номерЗаказа (code), имяПокупателя (firstName), фамилияПокупателя (lastName), телефон (phoneNumber), электронныйАдрес (e-mail), товар (item), суммаЗаказа (total).

Для класса Товар (Order) определим атрибуты артикул

(productCode), название (productName), размер (size), цвет (color), цена (price).

Замечание. Конечно, каждый заказ должен содержать хотя бы один товар, но может состоять нескольких товаров, о том, как это показать на диаграмме, мы поговорим в разделе «Определение спецификаций атрибутов класса».

Диаграмма классов с атрибутами показа на рисунке 76.

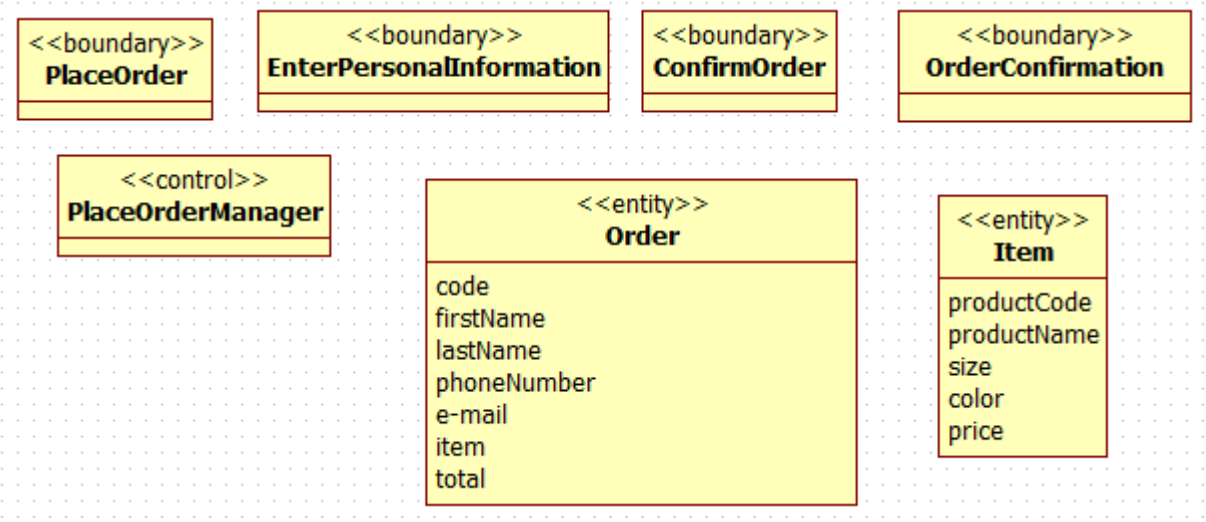


Рисунок 76. Диаграмма классов сценария Оформление заказа с атрибутами классов

Операция (operation) - это сервис, предоставляемый каждым экземпляром или объектом класса по требованию своих клиентов, в качестве которых могут выступать другие объекты, в том числе и экземпляры данного класса [1].

Пример. Класс ПодтверждениеЗаказа (OrderConfirmation) должен уметь отобразить информацию о заказе после его оформления, значит, он должен иметь соответствующую операцию: отобразитьЗаказ (displayOrder) (см. рис. 77).

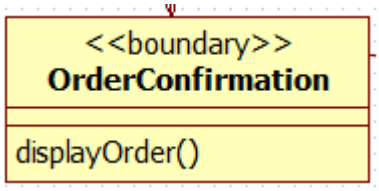


Рисунок 77. Операция класса

9.2 Как создать операцию класса в StarUML

Чтобы создать операцию класса в StarUML, щелкните один раз по этому классу, в редакторе свойств Properties откройте раздел Operations, нажав кнопку  . Откроется редактор коллекций (Collection Editor), в

котором нужно перейти в редактор операций, выбрав вкладку Operations (Операции).

Если в редакторе операций нажать кнопку , то будет создана новая операция и откроется редактор ее свойств, в котором можно изменить имя созданной операции (раздел Name) и задать другие спецификации (рис. 78).

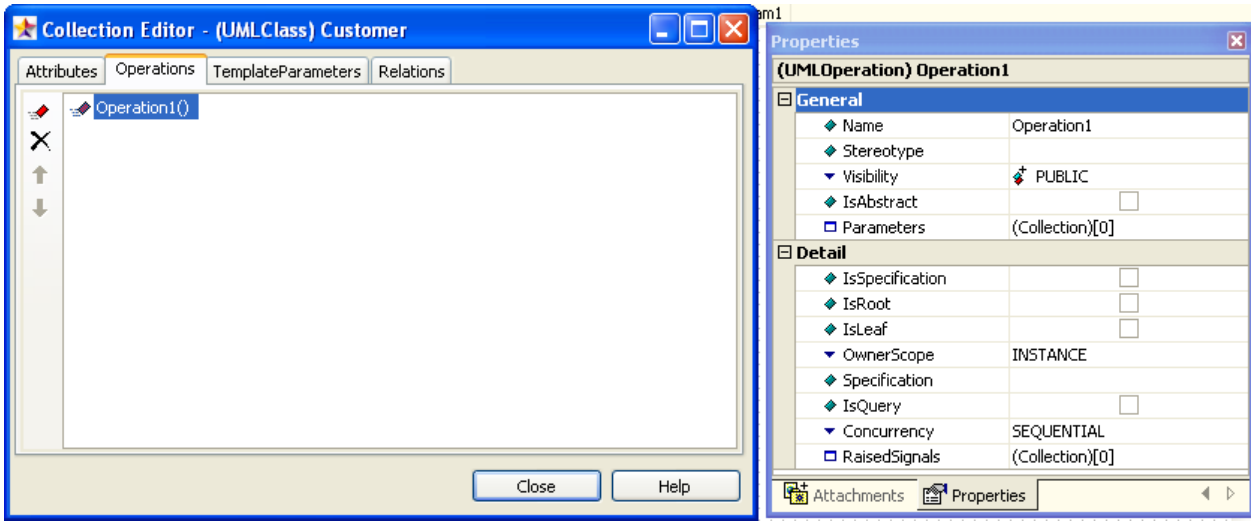


Рисунок 78. Создание операции класса

Для сохранения операции и ее спецификаций просто закройте диалог, нажав кнопку Close.

Более быстрый способ создать атрибут или операцию – это щелкнуть два раза левой кнопкой мыши по классу (рис. 79).

Рисунок 79. Быстрое создание атрибута

Нажав кнопку (для атрибута) или (для операции) вы получите атрибут или операцию соответственно.

Чтобы удалить атрибут или операцию щелкните по ней два раза левой кнопкой мыши и нажмите на значок справа. Чтобы добавить еще один атрибут или операцию нажмите на значок (рис. 80).

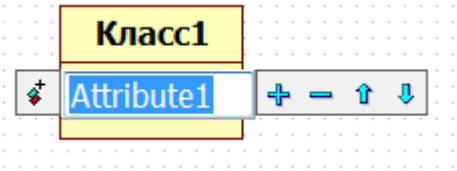


Рисунок 80. Быстрое добавление и удаление атрибута

9.3 Создание операций классов из сообщений на диаграмме последовательности

Сообщения, которые посылают объекты друг другу на диаграммах последовательности, как правило, вызывают определенную операцию класса этого объекта. Чтобы соотнести сообщение с операцией, нужно, чтобы каждому объекту был назначен классификатор, то есть класс, объектом которого он является.

Для того, чтобы соотнести объект с соответствующим ему классом, а классу присвоить объект, есть по крайней мере два способа.

- Щелкните два раза по объекту, для которого нужно создать класс. Затем щелкните на значок  справа от имени объекта и в появившееся окно введите имя класса. Класс будет автоматически создан в Logical View, а имя объекта изменится на составное. Перетащите созданный класс на соответствующую диаграмму классов. Заметим, что объекту был назначен его классификатор.
- Выделите объект, которому нужно назначить уже существующий классификатор. Справа на вкладке Properties (Свойства) выберите раздел Classifier (Классификатор), нажмите на значок  и в появившемся диалоговом окне найдите класс, соответствующий данному объекту.

Пример. После назначения классификаторов объектам диаграммы последовательности успешного сценария прецедента Оформить заказ внешний вид диаграммы изменится (см. рис. 81).

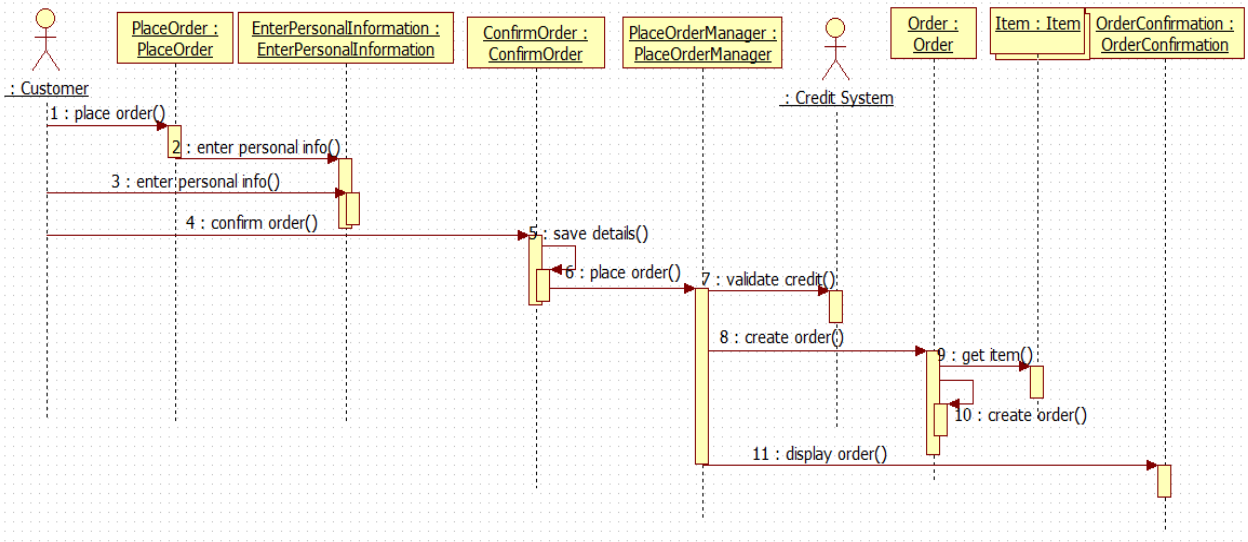


Рисунок 81. Измененная диаграмма последовательности сценария Оформить заказ

Замечание. Если мы создавали объекты на диаграмме последовательности с помощью перетаскивания классов с навигатора модели, то объекты автоматически связаны со своими классами.

В StarUML есть две возможности связать сообщение с операцией: можно создать операцию из сообщения, а можно использовать имя операции класса в качестве сообщения.

1. Для того чтобы создать операцию из сообщения, щелкните два раза по сообщению, нажмите на значок  справа от сообщения, и в открывшееся поле введите имя новой операции.

2. Для того чтобы использовать операцию класса как сообщение, щелкните два раза по сообщению, нажмите на значок  слева от сообщения, и в появившемся списке выберите нужную операцию.

Если подходить к понятию операции, как сервису, предоставляемому объектом другим объектам, то процедура создания операций из сообщений оказывается очень удобной для того, чтобы создавать операции классов.

Пример. На диаграмме последовательностей сценария Оформление заказа первое сообщение посылается объекту PlaceOrder. Этот объект должен уметь запускать оформление заказа, если корзина не пуста, создадим для него соответствующую операцию placeOrder.

Для класса EnterPersonalInformation создадим соответствующую операцию enterPersonalInformation из сообщения, посылаемого объектом PlaceOrder, а не из такого сообщения, посылаемого Покупателем – для покупателя это сообщение означает заполнение полей формы.

В класс OrderConfirmation добавим операцию displayOrder, создав ее из 9-ого сообщения «display order».

Создадим остальные операции классов, свяжем операции с сообщениями. На диаграмме последовательности вместо сообщений появятся операции (рис. 82).

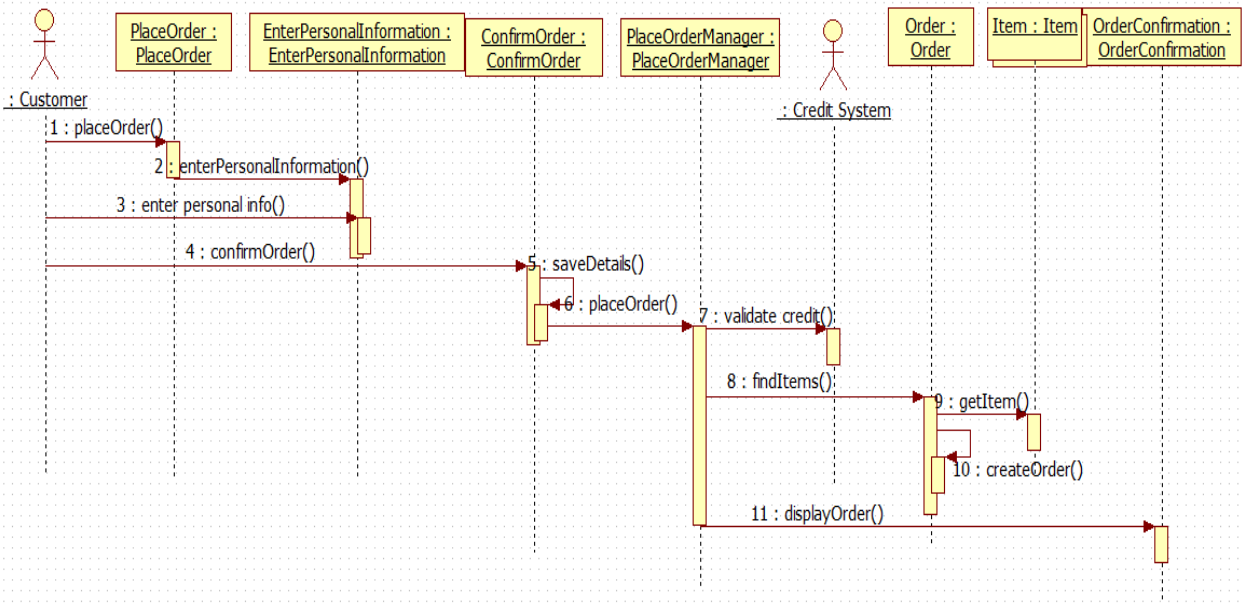


Рисунок 82. Диаграмма последовательности с операциями

А наша диаграмма классов с операциями, атрибутами и стереотипами будет выглядеть так, как показано ниже на рисунке 83.

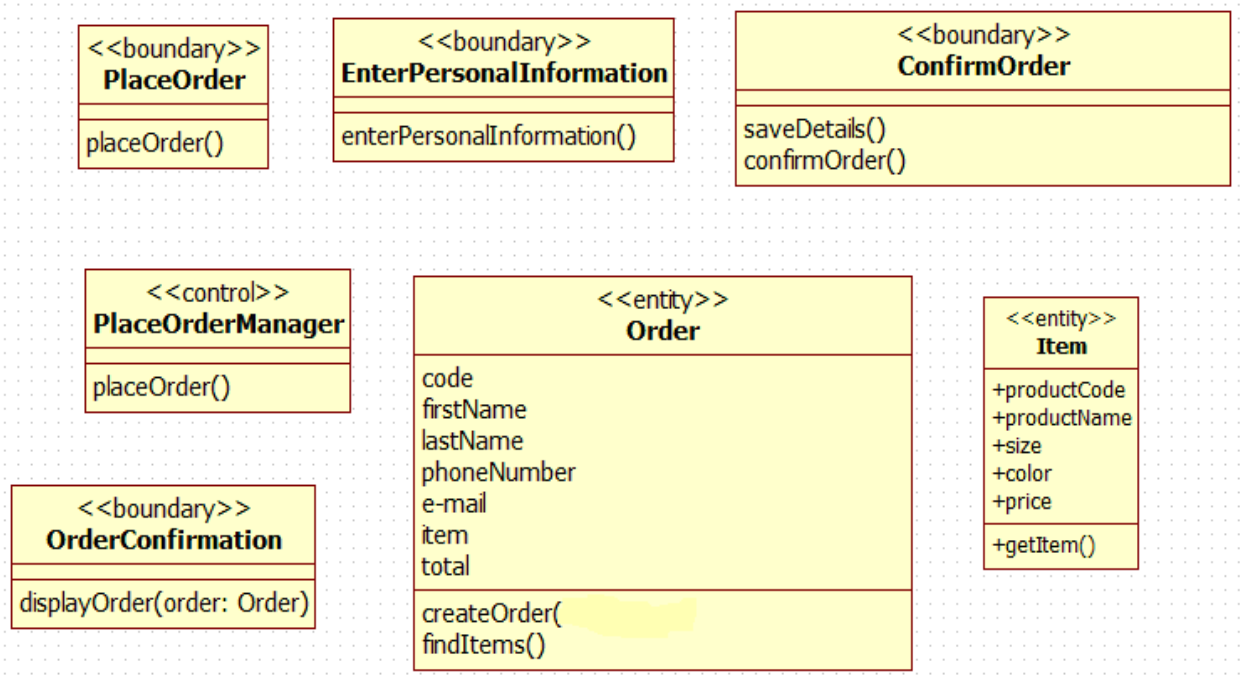


Рисунок 83. Диаграмма классов с операциями

10. Определение спецификаций атрибутов класса

В языке UML мы можем специфицировать атрибуты и операции заданием их видимости, кратности и пр.

Общий формат записи отдельного атрибута класса следующий:

[квантор видимости] имя атрибута [кратность] :
[тип атрибута] [= исходное значение] [{строка-свойство}]

Все элементы в квадратных скобках «[ ]» являются необязательными спецификациями атрибутов и могут быть опущены. Однако их использование позволяет сделать модель более полной и управлять взаимоотношениями между классами, разграничивая их права доступа.

Пример.

фамилия – указано только имя атрибута;
+фамилия – имя и видимость;
фамилия : String – имя и тип значений атрибута;
товаровВКорзине [0..] : Integer* – имя, кратность и тип;
-ID [1] : String {frozen} – видимость, имя, кратность, тип и свойство;
товаровВКорзине : Integer = 0 – имя и начальное значение.

Опишем спецификации атрибутов подробно.

Имя атрибута может быть произвольной текстовой строкой. Имя является единственным обязательным элементом при задании атрибута. Имя должно начинаться с маленькой буквы, если оно содержит несколько слов, то остальные слова, кроме первого, пишутся с большой буквы:

фамилия или *фамилияСотрудника*.

Видимость (visibility) — качественная характеристика описания свойств класса, характеризующая потенциальную возможность других объектов модели использовать это свойство (атрибут или операцию).

Видимость в языке UML обозначается с помощью квантора видимости *(visibility)*, который может принимать одно из 4-х возможных значений и отображаться при помощи специальных символов.

- Открытый (*public*). Атрибут виден всем остальным классам. Любой класс, связанный с данным в рамках диаграммы или пакета, может просмотреть или изменить значение атрибута. Обозначается символом «+» перед именем атрибута.
- Защищенный (*protected*). Любой потомок данного класса может пользоваться его защищенными свойствами. Обозначается знаком «#» перед именем атрибута.
- Закрытый (private). Атрибут с этой областью видимости недоступен или не виден для всех классов без исключения. Обозначается знаком «-» перед именем атрибута.
- Пакетный (package). Атрибут является открытым, но только в пределах своего пакета. В StarUML данный атрибут обозначается значком «~».

10.1 Определение видимости атрибута в StarUML

Когда мы создаем атрибут класса, StarUML автоматически делает его открытым. Квантор видимости изображается рядом с именем атрибута, слева. Чтобы изменить видимость, выделите атрибут, щелкнув по нему два раза, щелкните по появившемуся значку  слева от имени атрибута и в списке выберете желаемую видимость атрибута (рис. 84).

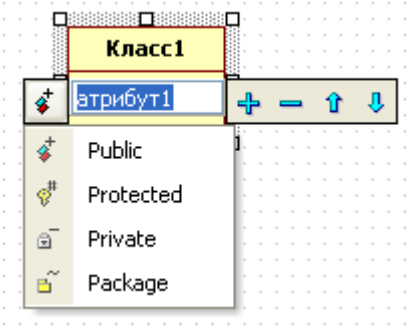


Рисунок 84. Определение видимости атрибута

Изменить видимость атрибута можно, также используя редактор свойств данного атрибута, раздел Visibility (Видимость) (рис. 85).

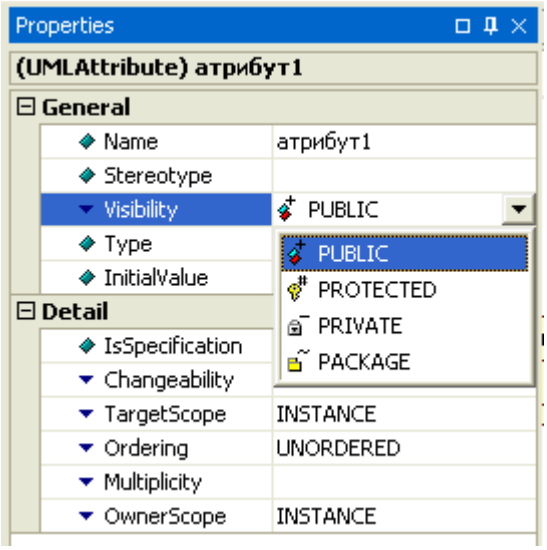


Рисунок 85. Определение видимости атрибута из редактора свойств

Пример. В классе Item (Товар) атрибут productCode сделаем защищенным. А в классе Заказ (Order) защищенным будет атрибут code (рис. 86).

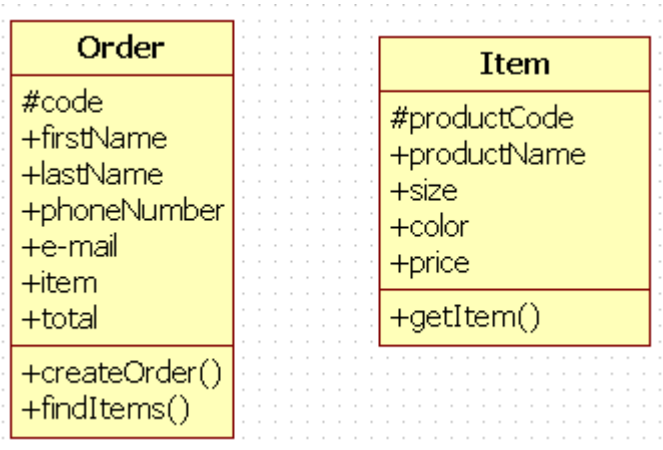


Рисунок 86. Атрибуты классов Товар и Заказ с видимостью

Квантор видимости может быть опущен. Его отсутствие означает, что видимость атрибута не указывается. Вместо условных графических

обозначений можно записывать соответствующее ключевое слово: public, protected, private, package или использовать значок StarUML для обозначения видимости.

Чтобы не отображать кванторы видимости на диаграмме, нужно выполнить следующие действия: щелкнуть правой кнопкой мыши по классу, в контекстном меню выбрать пункт Format, затем выбрать Show Compartment Vizibility (рис. 87)

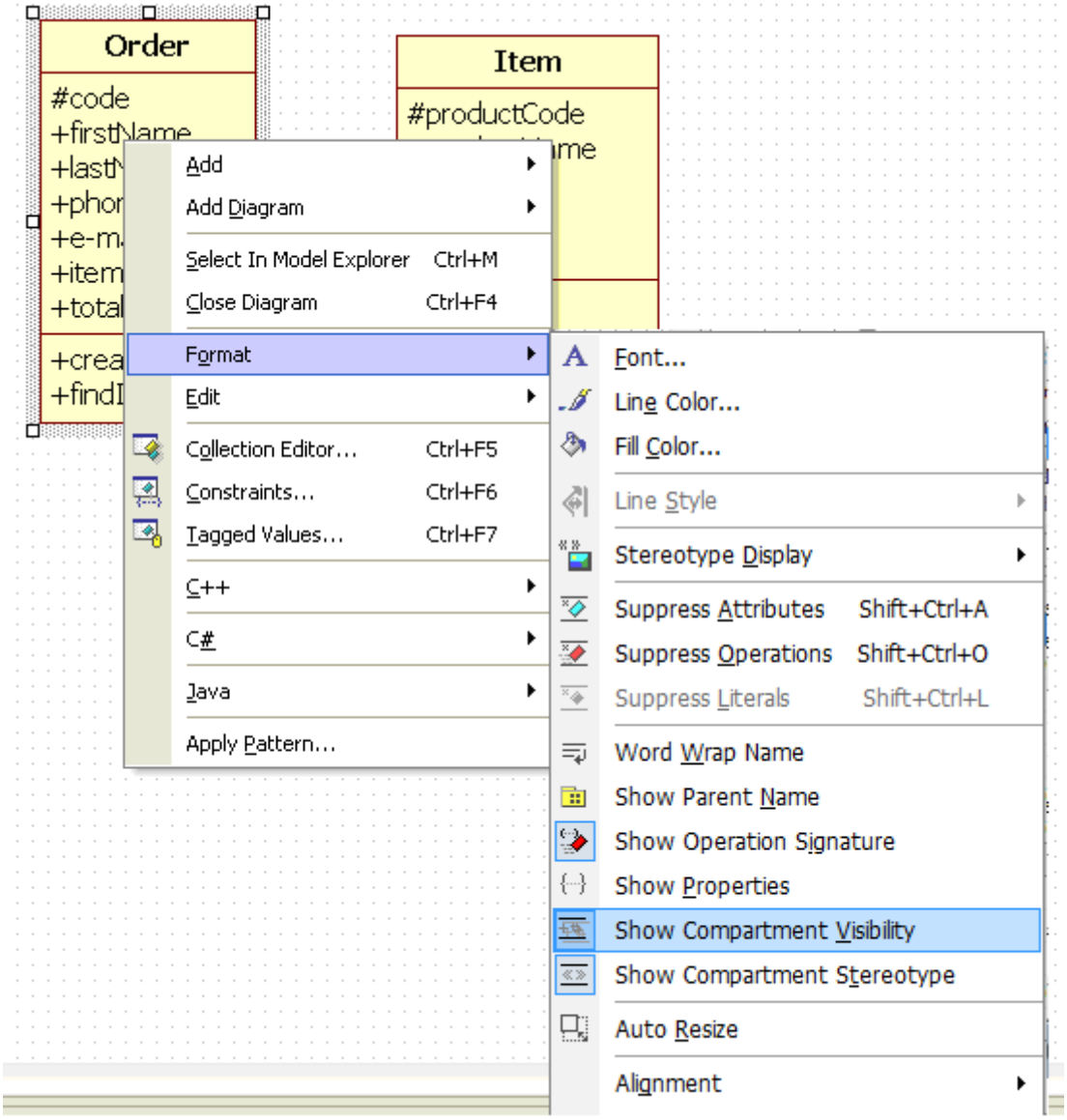


Рисунок 87. Отображение кванторов видимости

Кратность (multiplicity) — спецификация области значений допустимой мощности, которой могут обладать соответствующие множества.

Кратность указывает, сколько экземпляров данного атрибута может иметь экземпляр класса. Значение кратности записывается в квадратных скобках, в которых указывается возможный диапазон кратности атрибута:

[нижняя граница .. верхняя граница]

где нижняя и верхняя границы положительные целые числа. В качестве верхней границы может использоваться специальный символ « *» (звездочка), который означает произвольное положительное целое число, т.е. неограниченное сверху значение кратности соответствующего атрибута.

Интервалов кратности отдельного атрибута может быть несколько. 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. Ниже приведены некоторые примеры записи кратности атрибута.

Пример.

0..1	ноль или один;
1 или 1..1	ровно один;
2..*	два или больше;
2..5	2,3,4 или 5
1..3,5,8..10	1,2,3,5,8,9или 10
*	любое положительное число или нуль

10.2 Определение кратности атрибута в StarUML

Для задания кратности атрибута в StarUML нужно найти атрибут, открыв раздел Attributes из редактора свойств соответствующего класса, выделить атрибут, откроется его редактор свойств и в разделе Multiplisity выбрать кратность (рис. 88).

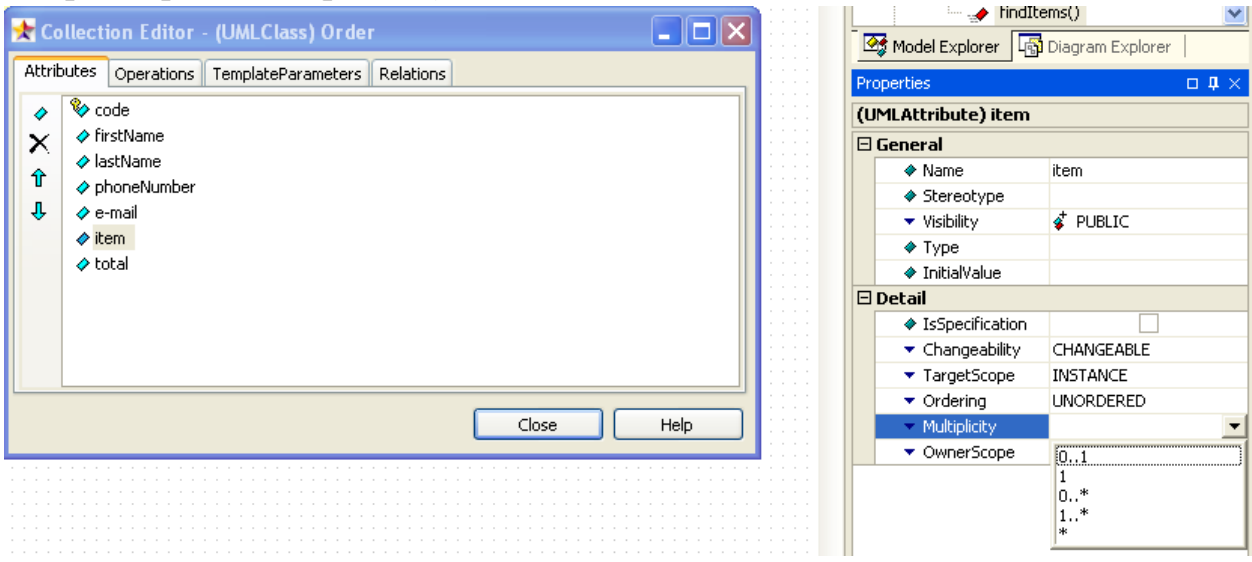


Рисунок 88. Задание кратности атрибута

Пример. Как уже отмечалось ранее, каждый заказ должен содержать хотя бы один товар, но может состоять нескольких товаров. Чтобы позволить экземпляру класса Заказ (Order) иметь несколько экземпляров атрибута товар (item), определим кратность этого атрибута как [1..*] (рис. 89).

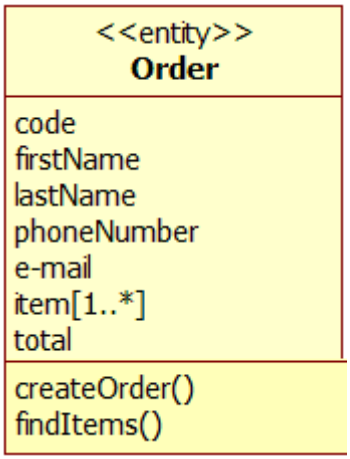


Рисунок 89. Атрибут item с кратностью

Тип атрибута представляет собой выражение, семантика которого определяется некоторым типом данных, определенным в пакете «Типы данных языка UML» или самим разработчиком. Тип атрибута может определяется в зависимости от языка программирования, который предполагается использовать для реализации данной модели. Если в качестве атрибута класса выступает другой класс, то типом атрибута будет этот класс. В простейшем случае тип атрибута указывается строкой текста, имеющей осмысленное значение в пределах пакета или модели, к которым относится рассматриваемый класс.

10.3 Определение типа атрибута в StarUML

Для задания типа атрибута в StarUML нужно найти атрибут, открыв раздел Attributes из редактора свойств соответствующего класса, выделить атрибут, откроется его редактор свойств, открыть в нем раздел Type, нажав ... В появившемся диалоговом окне выберите один из стандартных типов, либо один из определенных вами типов (классов) (рис. 90).

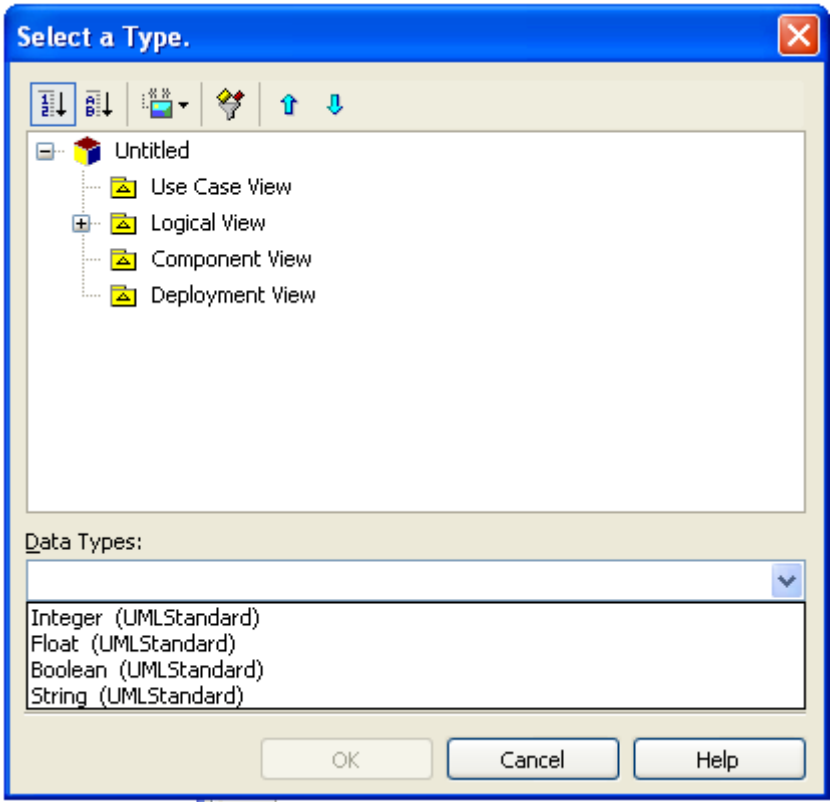


Рисунок 90. Определение типа атрибута

Пример. Атрибут item класса Заказ (Order) должен быть экземпляром класса Товар (Item). Для остальных атрибутов зададим стандартные типы (рис. 91).

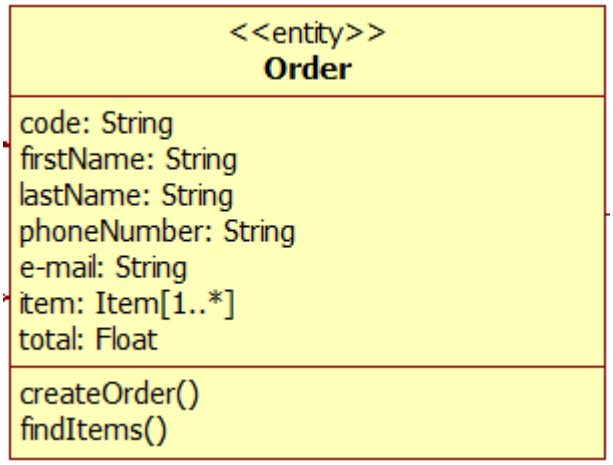


Рисунок 91. Типы атрибутов класса Order

Исходное значение служит для задания начального значения 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 объекта может переопределять исходное значение в процессе выполнения программы, если в этом возникает необходимость.

Строка-свойство служит для указания дополнительных свойств атрибута, которые могут характеризовать особенности изменения значений атрибута в ходе выполнения программы. 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.

11. Определение спецификаций операций класса

Поведенческие характеристики класса моделируются определением операций класса. На первом уровне абстракции достаточно просто записать их имена, но для операций, также как и для атрибутов, определен ряд спецификаций. Наиболее полный синтаксис записи операции в UML следующий:

[квантор видимости] имя операции [(список параметров)]
[: выражение типа возвращаемого значения] [{строка-свойство}]

Все элементы, стоящие в квадратных скобках «[]», являются необязательными спецификациями операций, однако наличие круглых скобок в описании операции обязательно, даже если список параметров пуст.

Пример.

отобразить() – указано только имя операции;

+отобразить() – имя и видимость;

+добавитьТоварВКорзину(inout t:Товар) – видимость, имя, параметр, тип параметра и его направление;

удалитьТоварИзКорзины(q:ТоварВКорзине) – указаны имя и параметр;

изменитьКоличествоТовара(q:Товар в корзине, inout n: Integer): Integer – имя операции, параметры, направление параметра и тип возвращаемого значения.

Раскроем смысл спецификаций операций.

Имя операции совместно с ее параметрами называют ***сигнатурой*** операции. Имя операции – это строка текста, обычно используют глагол или

короткое глагольное выражение, если оно состоит из нескольких слов, то все слова, кроме первого, пишутся с большой буквы:

добавить или *добавитьТоварВКорзину*.

Так же как и для атрибутов класса, ***видимость*** (visibility) операции обозначается с помощью квантора видимости и имеет четыре допустимых значения:

- Открытый (*public*). Атрибут виден всем остальным классам. Любой класс, связанный с данным в рамках диаграммы или пакета, может просмотреть или изменить значение атрибута. Обозначается символом «+» перед именем атрибута.
- Защищенный (*protected*). Любой потомок данного класса может пользоваться его защищенными свойствами. Обозначается знаком «#» перед именем атрибута.
- Закрытый (*private*). Атрибут с этой областью видимости недоступен или не виден для всех классов без исключения. Обозначается знаком «-» перед именем атрибута.
- Пакетный *(package*). Атрибут является открытым, но только в пределах своего пакета. В StarUML данный атрибут обозначается значком «~».

Квантор видимости для операции может быть опущен. В этом случае его отсутствие просто означает, что видимость операции не указывается. Вместо условных графических обозначений также можно записывать соответствующее ключевое слово: public, protected, private, package.

Квантор видимости атрибутов и операций также может быть указан в виде специального значка или символа, которые используются для графического представления моделей в инструментальном средстве.

Пример. Для класса Товар (Item) мы определили операцию getItem(), которая должна быть доступна классу Заказ (Order) для вызова, остальным классам знать об этой операции не обязательно. Сделаем ее защищенной (рис. 92).

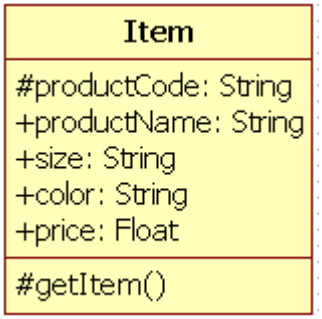


Рисунок 92. Защищенная операция класса

Сигнатура операции может содержать ноль или более ***параметров***, каждый из которых имеет следующий синтаксис:

[направление параметра] имя параметра : тип параметра [= значение параметра по умолчанию]

Имя параметра есть идентификатор соответствующего формального параметра, при записи которого следуют правилам задания имен.

Тип параметра является спецификацией типа данных для допустимых значений соответствующего формального параметра. Тип данных параметра может быть стандартным типом UML, либо, если в качестве входного параметра операция использует целый класс, то параметр имеет типом этот класс. В StarUML допускаются четыре стандартных типа UML: String, Integer, Float, Boolean. Параметр может также быть типом класса, который используется в данной операции. Тип параметра записывается с большой буквы.

11.1 Определение параметров операции в StarUML

Для задания параметра операции в StarUML нужно найти атрибут, открыв раздел Operations из редактора свойств соответствующего класса, выделить операцию, откроется ее редактор свойств, открыть в нем раздел Parameters, нажав ... . В появившемся диалоговом окне, чтобы создать параметр, нажмите значок . Параметр будет создан со стандартным именем Parameter1. Чтобы удалить параметр, используйте кнопку . При создании параметра откроется его редактор свойств, в котором можно изменить имя параметра и определить его тип, открыв раздел Type (рис. 93).

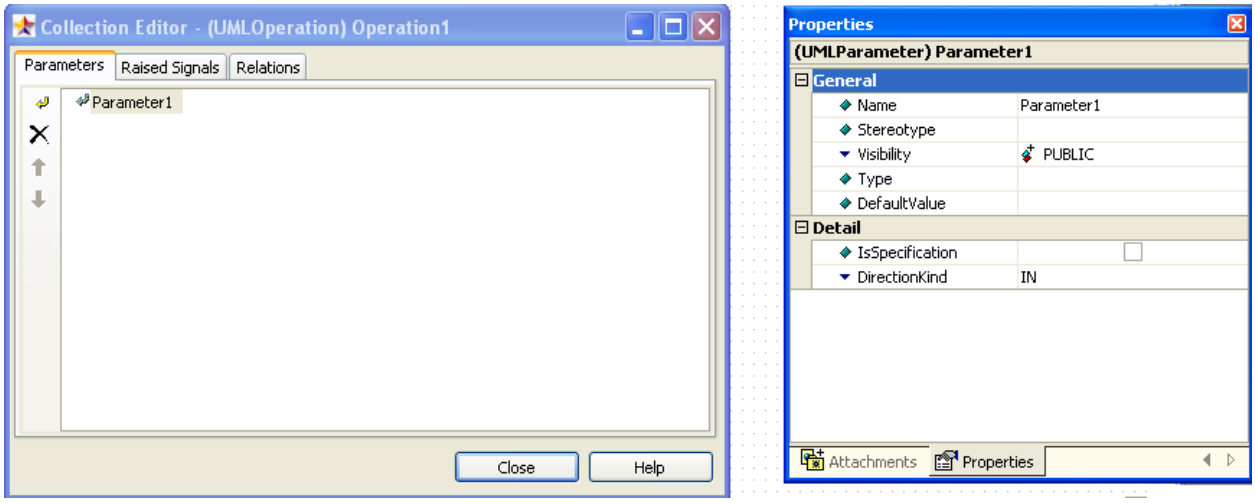


Рисунок 93. Добавление параметра операции

Пример. В нашем примере Оформление заказа чтобы создать

подтверждение заказа и показать его на экране, классу OrderConfirmation (ПодтверждениеЗаказа) нужна информация о заказе. Для того чтобы OrderConfirmation (ПодтверждениеЗаказа) узнавал об изменениях в классе Order (Заказ) нужно в OrderConfirmation (ПодтверждениеЗаказа) использовать Order (Заказ) в качестве параметра операции (рис. 94).

Рисунок 94. Параметр операции типа Order

Направление параметра может принимать одно из нижеследующих значений:

- *in* – входящий параметр, который не может быть изменен;
- *out* – выходящий параметр, который может быть изменен, чтобы передать информацию вызвавшей процедуре;
- *inout* – входящий параметр, который может быть изменен.

по умолчанию, если направление параметра не указано, принимается значение in.

Значение параметра по умолчанию в общем случае представляет собой некоторое конкретное значение для этого формального параметра.

Выражение типа возвращаемого значения операции также указывает на тип данных значения, которое возвращается объектом после выполнения соответствующей операции. Оно может быть опущено, если операция не возвращает никакого значения. Для указания нескольких возвращаемых значений данный элемент спецификации операции может быть записан в виде списка отдельных выражений.

Выражение типа возвращаемого значения еще называют возвращаемым классом операции. Для определения возвращаемого класса можно использовать встроенные типы (string, float, integer, boolean) или типы, определенные в вашей модели.

Пример. Для того чтобы класс Order (Заказ) мог найти товары и добавить их в заказ, мы определили для него операцию findItem(). Результатом выполнения этой операции может быть элемент типа item (рис. 95).

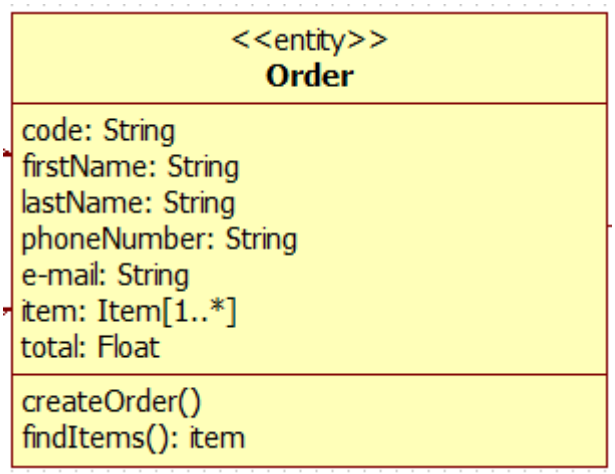


Рисунок 95. Возвращаемое значение типа item

Строка-свойство в описании операции служит для указания значений свойств, которые могут быть применены к данной операции. Строка-свойство может отсутствовать, если свойства не специфицированы.

12. Отношения между классами

Отношение - это связь между классами.

Для того чтобы обнаружить связи классов исследуются сценарии и диаграммы последовательности: если объект посылает сообщение другому объекту, то по-видимому, между ними существует отношение.

Отношения возникают также, если один класс использует другой в качестве параметра операции.

В нотации UML определены 5 видов отношений между классами:

- ассоциации;
- агрегации;
- композиции;
- зависимости;
- обобщения.

В нотации UML отношение между классами изображается на диаграммах стрелками, вид стрелки зависит от семантики отношения.

12.1 Создание отношения между классами в StarUML

Чтобы создать отношение между классами в StarUML щелкните один раз по обозначению элемента связи на панели элементов слева, выбав тот тип отношения, который вам нужен, а затем проведите линию от одного класса к другому, удерживая левую кнопку мыши (рис. 96).

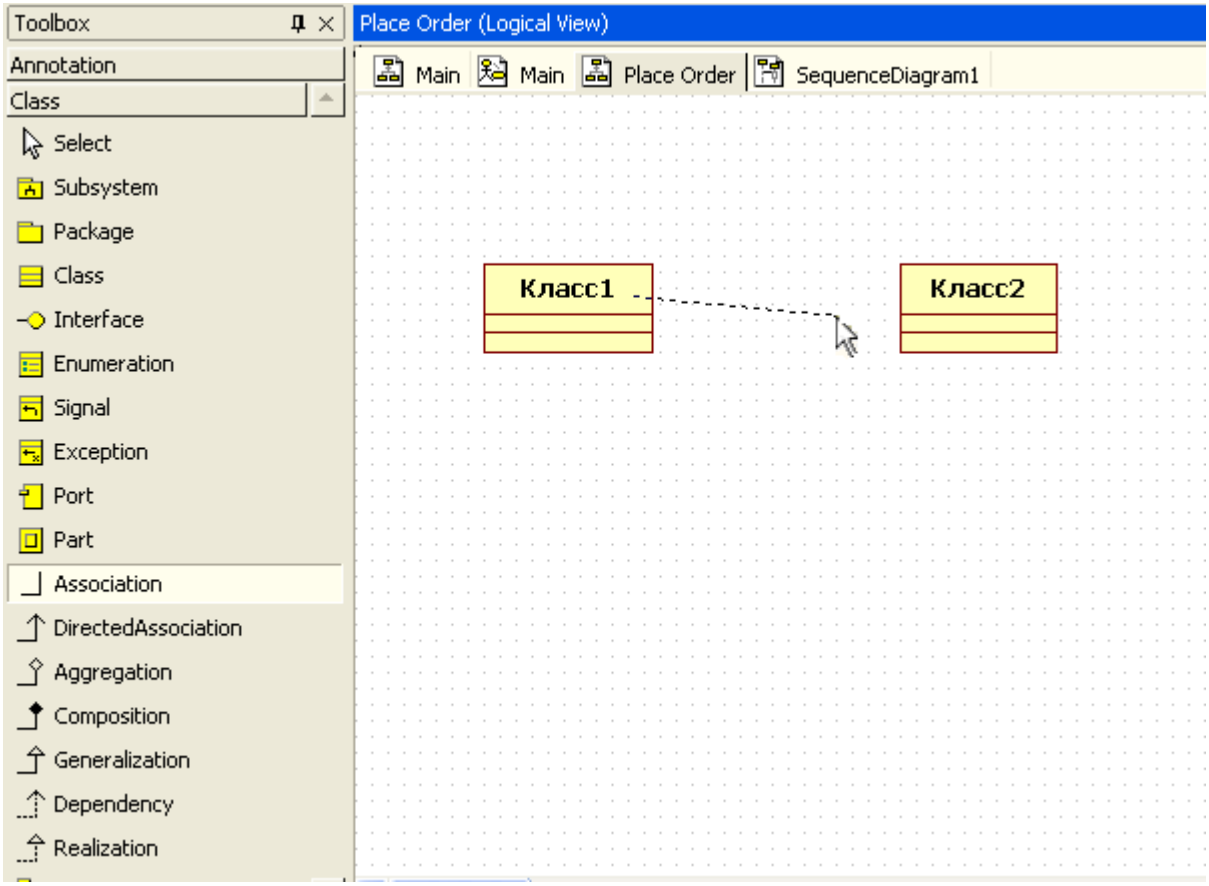


Рисунок 96. Создание отношения между классами

Ассоциация – это семантическая связь между классами. На диаграмме ее рисуют в виде обычной линии. Ассоциация может быть однонаправленной или двунаправленной. В первом случае ее изображают в виде стрелки, показывающей направление связи. Во втором случае – двойной стрелки либо просто линии без стрелок.

Если между классами создана двунаправленная связь, то каждый из них видит открытые атрибуты и операции других классов (рис. 97).

Пример.

Рисунок 97. Двунаправленная ассоциация

Если между классами установлена однонаправленная ассоциация, то в этом случае класс, от которого направлена стрелка, знает открытые атрибуты и операции второго класса, а второй класс, к которому идет стрелка, не видит атрибуты и операции первого класса. Например, класс Покупатель знает название и размерный ряд Товара, но Товар не знает имя и адрес Покупателя (рис. 98).

Пример.

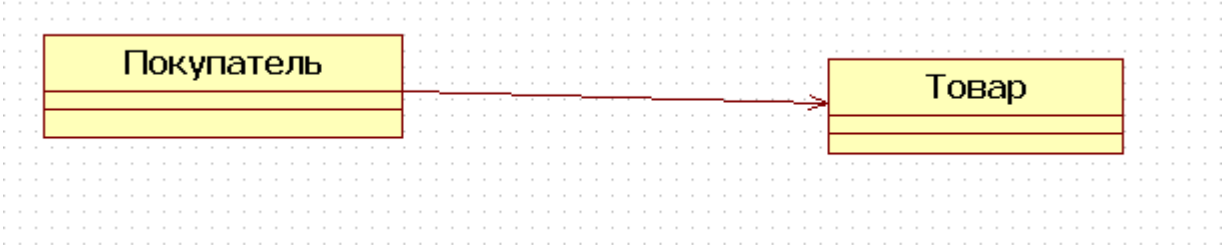


Рисунок 98. Направленная ассоциация

При таком отношении между классами Покупатель должен знать о классе Товар и потому не может использоваться без него, но класс Товар не зависит от Покупателя и может быть повторно использован самостоятельно, независимо от класса Покупатель.

Ассоциация может быть рефлексивной, это означает, что один экземпляр класса обращается к другому экземпляру этого класса. Изображается рефлексивное отношение как на рисунке 99.

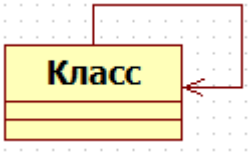


Рисунок 99. Рефлексивная ассоциация

К ассоциации применимы четыре базовых дополнения: имя, роль, кратность и агрегирование.

Ассоциации можно присвоить ***имя***. Обычно в качестве имени используется глагол или предложение, поясняющее причину возникновения связи (рис. 100).

Пример.

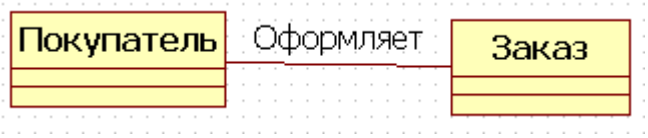


Рисунок 100. Имя ассоциации

Окончание линии ассоциации в месте, где она соединяется с классом, называется ***ролью ассоциации***.

Название роли ассоциации – это обычно существительное, которое уточняет, описывает роль, в которой один класс участвует в связи с другим классом. Роль может быть указана для обоих классов, участвующих в связи, либо только для одного (рис. 101).

Пример.

Рисунок 101. Ассоциация с ролями

Кратность (мощность) определяется для классов и указывает допустимое количество объектов (экземпляров класса), участвующих в отношении.

Кратность указывает, сколько экземпляров одного класса взаимодействуют с помощью отношения с одним экземпляром данного класса в данный момент.

Примеры индикаторов мощности:

0..1	ноль или один;
1	ровно один;
1..*	один или много;
2..5	2,3,4 или 5
6..8,10	6,7,8 или 10

Пример. Покупатель может оформить много заказов или не оформить ни одного. Заказ должен быть сделан только 1 покупателем (рис. 102).

Рисунок 102. Кратность классов в ассоциации

Как видно из примера, читать кратность класса нужно на противоположном конце связи.

Указывать имя, роль или кратность связи необязательно. Это нужно делать, когда это может помочь более точному представлению модели системы и лучшему ее пониманию.

Агрегация - специальная форма ассоциации, которая служит для представления отношения типа "часть-целое" между агрегатом (целое) и его составной частью (рис. 103).

Рисунок 103. Агрегация

Графически агрегация изображается линией, на одном конце которой, принадлежащем целому, помещен не закрашенный ромб.

Пример. Любой заказ состоит из товаров (рис. 104).

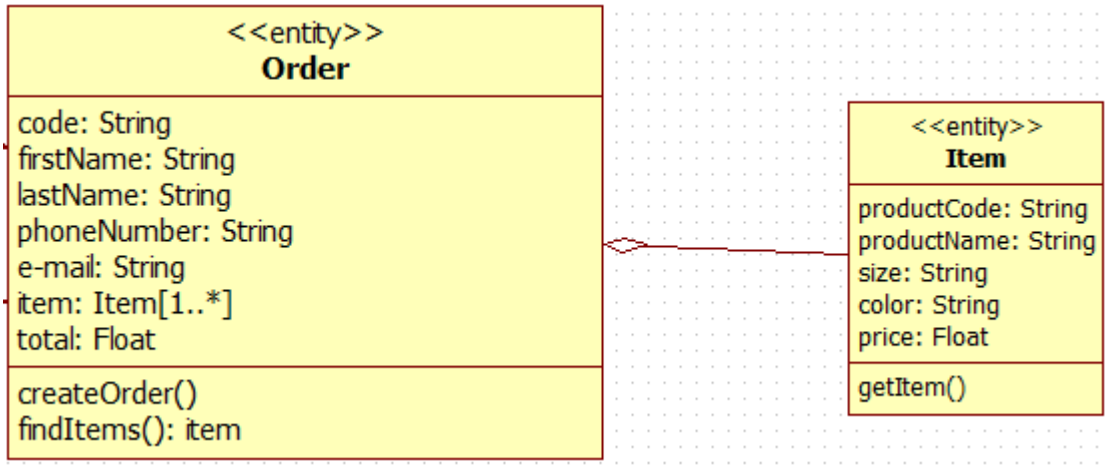


Рисунок 104. Агрегация между классами Товар и Заказ

Как и ассоциации, агрегации могут быть рефлексивными. Это означает, что один экземпляр класса состоит из одного или нескольких экземпляров того же класса.

Композицией называется форма агрегирования с четко выраженным отношением владения, причем время жизни частей и целого совпадают [1].

Как только будет уничтожен объект Целое, так вместе с ним будет уничтожен объект Часть.

На диаграммах композиция показывается так же, как и агрегация, но только ромб должен быть закрашен.

Пример. Классический пример: если открыто окно программы, то мы видим доступные нам кнопки, например, кнопка «Добавить» (товар). Как только мы закроем окно программы, кнопки перестанут существовать (рис.105).

Рисунок 105. Композиция

Так как агрегация и композиция являются отношениями ассоциации, то для них допустимо, но не обязательно, указывать имя, роль и кратность.

Зависимостью называется отношение использования, определяющее, что изменение в спецификации одной сущности может повлиять на другую

сущность, которая ее использует, причем обратное в общем случае не верно [1].

Графически зависимость изображается в виде пунктирной стрелки, которая идет к той сущности, от которой зависит еще одна. Зависимости применяются, чтобы показать, что один класс использует другой. Т.е. один класс является клиентом другого класса-поставщика и использует этот класс-поставщик как параметр своей операции.

Пример. Мы используем класс Order (Заказ) как входной параметр операции displayOrder (отобразитьЗаказ) класса OrderConfirmation (ПодтверждениеЗаказа). Так как для выполнения этой операции, класс OrderConfirmation (ПодтверждениеЗаказа) использует класс Order (Заказ), то они связаны отношением зависимости (рис. 106).

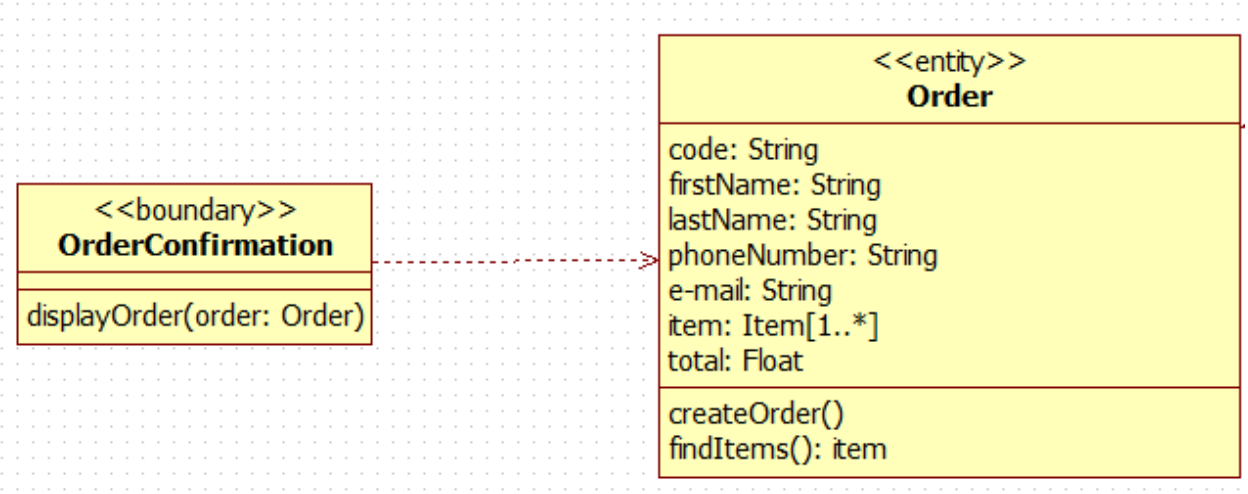


Рисунок 106. Зависимость между классами

Создание экземпляра класса ПодтверждениеЗаказа не повлечет за собой создание экземпляра класса Заказ. Однако эти два класса смогут обмениваться сообщениями на диаграммах взаимодействия.

Обобщение – это отношение наследования между двумя элементами модели. Оно дает классу возможность наследовать открытые или защищенные атрибуты и операции суперкласса (класса от которого наследуются атрибуты и операции). Помимо наследуемых каждый класс может иметь свои атрибуты и операции.

На диаграммах обобщение изображается в виде стрелки с не закрашенным треугольником у суперкласса, идущей от потомка.

Пример. В магазине могут работать различные сотрудники: сотрудник отдела продаж, кладовщик, директор. Все они имеют общие свойства: имя, адрес, телефон, дата рождения, должность, поэтому можно рассматривать обобщающую сущность Сотрудник, атрибуты и операции которой сущности Директор и Кладовщик будут наследовать (рис. 107).

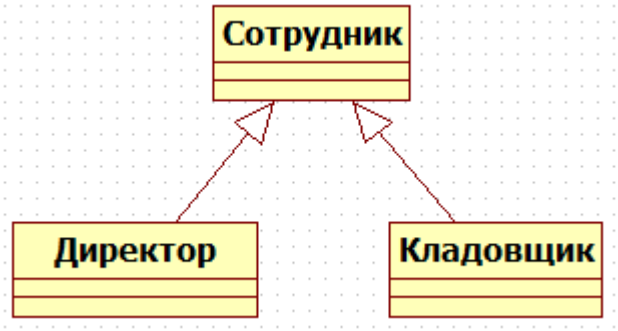


Рисунок 107. Отношение обобщения между классами

Если Сотрудник имеет в качестве атрибутов имя, адрес, телефон, дату рождения, должность, то сущности, Директор и Кладовщик, конечно, наследуют эти атрибуты со своими значениями. Кроме того они могут иметь и собственные атрибуты или операции. Например, у Директора может быть операция уволитьСотрудника, которой не может быть у Кладовщика, а у последнего операция - выдатьТовар.

Закрытые атрибуты и операции **не могут** наследоваться потомками.

Пример. Определим отношения между классами сценария Оформление заказа (рис. 108).

Проанализировав диаграмму последовательности выясняем, что класс PlaceOrder связан с EnterPersonalInformation, а объект ConfirmOrder посылает сообщения объекту класса PlaceOrderManager. PlaceOrderManager связан с объектами классов Order и OrderConfirmation. Для всех перечисленных связей определим отношения ассоциации.

Класс OrderConfirmation использует класс Order как параметр своей операции: между ними определим отношение зависимости.

Экземпляры класса Order состоят из экземпляров класса Item. Между ними создадим отношение агрегации.

Для того чтобы класс ConfirmOrder мог выполнять операцию подтверждения заказа, он должен быть связан с классом EnterPersonalInformation, поэтому создадим между ними отношение ассоциации.

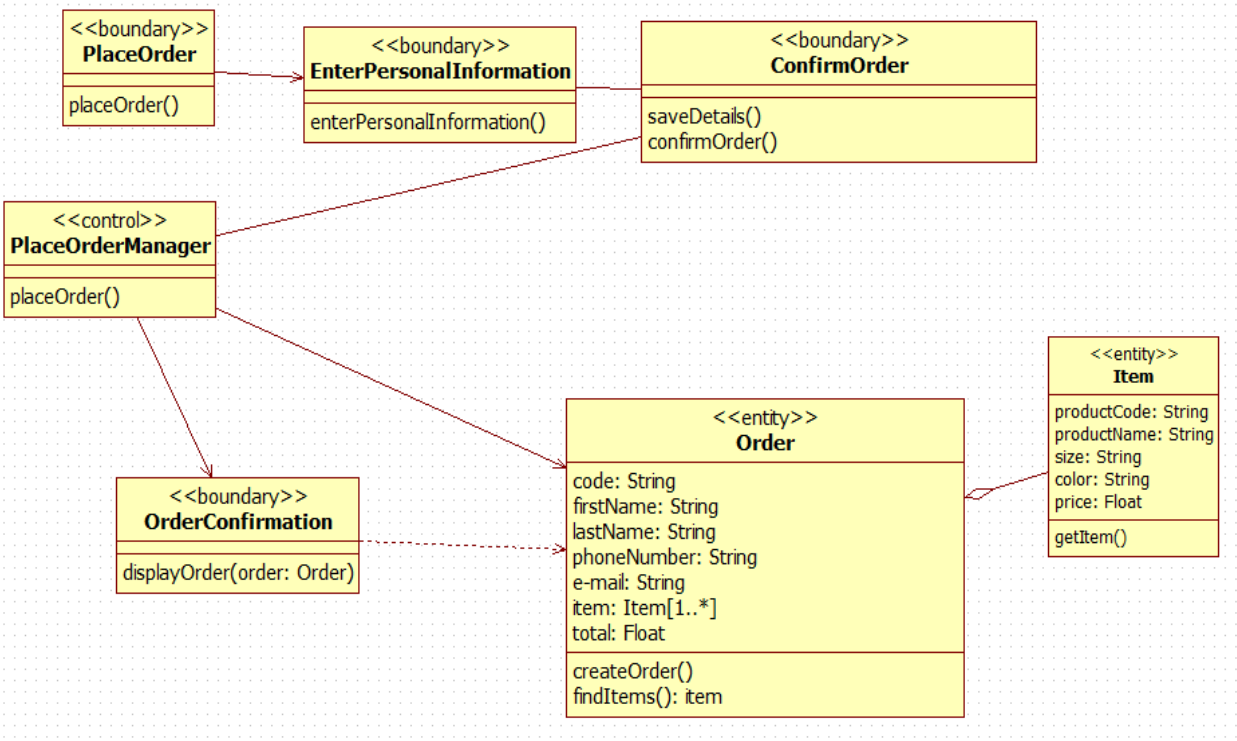


Рисунок 108. Диаграмма классов с отношениями

12.2 Отношения между пакетами

Отношения между пакетами могут быть только отношением типа зависимости. Обозначается такое отношение пунктирной стрелкой (рис. 109).

Рисунок 109. Зависимость между пакетами

Если Пакет1 зависит от Пакета2, как это изображено на рисунке, то это значит, что один или несколько классов Пакета1 инициируют связь с общедоступными классами Пакета2. Пакет1 называется пакетом-клиентом, Пакет2 называется пакетом-поставщиком.

Метод создания связей между пакетами в StarUML такой же, как и между классами.

13. Диаграммы состояний

Для представления взаимодействий объектов в системе используют диаграммы последовательности и кооперации. Прецеденты и сценарии описывают поведение системы, но если нам важна динамика класса, изменение состояний отдельного объекта, то для этой цели полезно

построить диаграммы состояний.

Диаграмма состояний показывает положение одиночного объекта, события и сообщения, которые вызывают переход из одного состояния в другое, и действия, являющиеся результатом смены состояния. Диаграмма состояний показывает объект с момента его создания и до его уничтожения.

Диаграмму состояний строят не для каждого класса в системе, а только для классов с динамическим поведением, которые отсылают и принимают много сообщений, изменяют свое состояние. Программный код из диаграмм состояний не генерируются, но они важны для понимания динамики поведения класса, дают возможность понять логику изменений перед кодированием.

13.1 Создание диаграммы состояний в StarUML

Для добавления диаграммы состояний в модель нужно выполнить следующие шаги: щелкнуть правой кнопкой мыши по папке представления Logical View в навигаторе модели, в контекстном меню выбрать пункт Add Diagram, в списке выбрать диаграмму состояний Statechart Diagram (рис. 110).

Мы также можем связать диаграмму состояний с тем классом, состояния объекта которого она описывает. Для этого нужно щелкнуть правой мышкой по соответствующему классу, а не по папке Logical View.

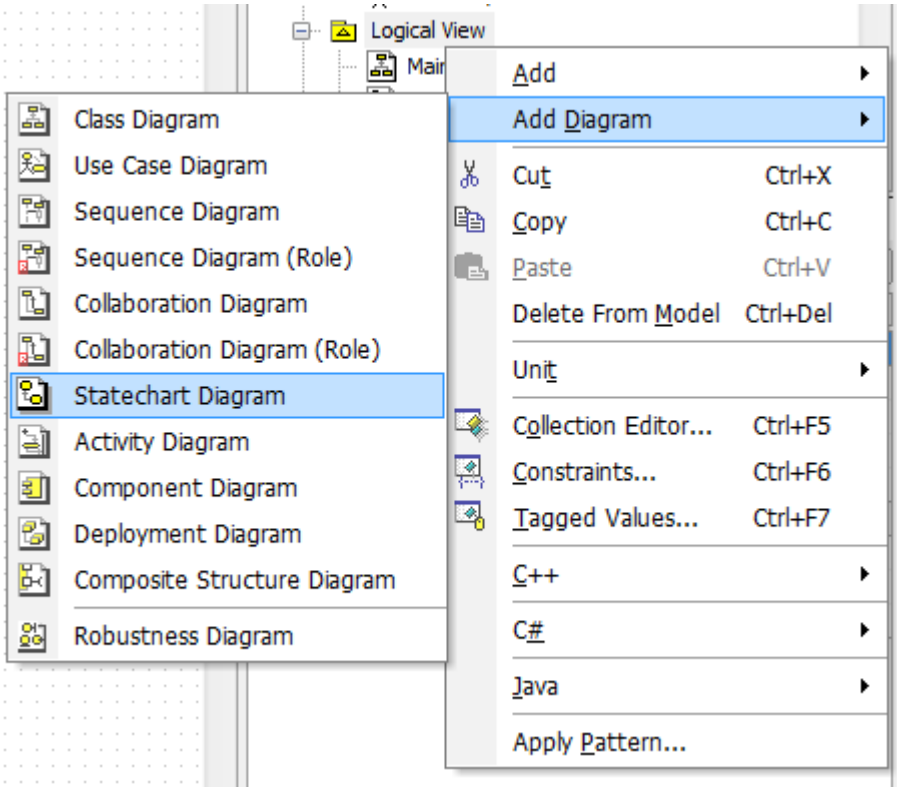


Рисунок 110. Добавление диаграммы состояний

13.2 Основные элементы диаграмм состояний

Диаграмму состояний часто рассматривают в контексте конечного автомата. Тогда можем сказать, что ***диаграмма состояний (Statechart diagram)*** показывает автомат, фокусируя внимание на потоке управления от состояния к состоянию [2].

Автомат (State machine) – это описание последовательности состояний, через которые проходит объект на протяжении всего жизненного цикла, реагируя на события, - в том числе описание реакций на эти события.

Состояние (State) – это ситуация в жизни объекта, на протяжении которой он удовлетворяет некоторому условию, осуществляет определенную деятельность или ожидает какого-то события.

Для поиска состояний класса можно просматривать атрибуты этого класса. Хорошим индикатором состояний является такой атрибут класса как «статус».

Диаграмма состояний изображается в виде графа с вершинами и ребрами.

Состояние на диаграмме изображается прямоугольником со скругленными вершинами. Под именем состояния могут размещаться действия (рис. 111).

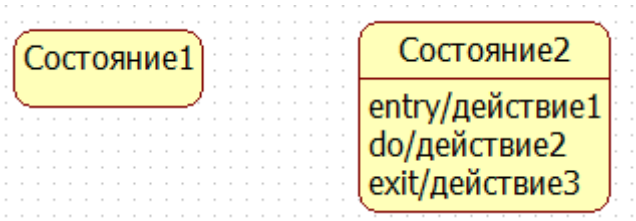


Рисунок 111. Состояния

Кроме обычных состояний на диаграмме состояний могут размещаться псевдосостояния.

Псевдосостояние (pseudo-state) - вершина в конечном автомате, которая имеет форму состояния, но не обладает поведением.

Примерами псевдосостояний, которые определены в языке UML, являются начальное и конечное состояния.

Начальное состояние (start state) - разновидность псевдосостояния, обозначающее начало выполнения процесса изменения состояний конечного автомата или нахождения моделируемого объекта в составном состоянии.

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, из которого может только выходить стрелка-переход (рис. 112).

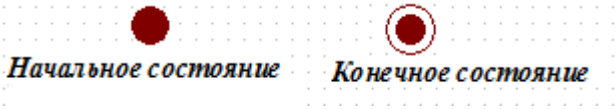


Рисунок 112. Начальное и конечное состояния

Конечное состояние (final state) - разновидность псевдосостояния, обозначающее прекращение процесса изменения состояний конечного автомата (рис. 112).

Объект находится в конечном состоянии непосредственно перед уничтожением. Конечных состояний может быть несколько.

Пример. Вернемся к нашему примеру магазина «Style». Покупатель оформляет заказ. Класс Заказ, кроме прочих атрибутов имеет атрибут «статус». Проследим динамику движения заказов в системе с помощью диаграммы состояний, составленной для класса Заказ.

По условию нашей задачи данные о сделанном заказе поступают сотруднику отдела продаж, который проверяет оплату, реквизиты заказа и передает его кладовщику на комплектацию. Кладовщик, проверив наличие заказанных товаров и собрав заказ, если это возможно, делает отметку о готовности.

Заказ выдается со склада кладовщиком. Кладовщик выдает заказ и отмечает в системе, что заказ выдан. Далее данные о заказе мы можем передать в архив.

Отразим на диаграмме переход заказа между состояниями (рис. 113).

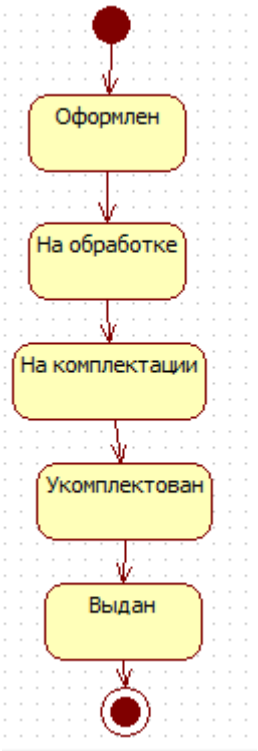


Рисунок 113. Диаграмма состояний объекта класса Заказ

Находясь в каком-либо состоянии, объект может выполнять определенные действия, с состоянием можно связать такие действия как входное и выходное действия и внутренняя или просто деятельность.

Действие Действие (Action) – это одиночное вычисление, которое приводит к смене состояния или возврату значения.

В общем случае действие имеет следующий формат:

<метка действия / выражение действия>

Входное действие (entry action) - действие, которое выполняется в момент перехода в данное состояние.

Обозначается с помощью ключевого слова - метки действия entry, которое указывает на то, что следующее за ней выражение действия должно быть выполнено в момент входа в данное состояние.

Действие выхода (exit action) - действие, производимое при выходе из данного состояния.

Обозначается с помощью ключевого слова - метки действия exit, которое указывает на то, что следующее за ней выражение действия должно быть выполнено в момент выхода из данного состояния.

Внутренняя деятельность (do activity) - выполнение объектом операций или процедур, которые требуют определенного времени.

Другими словами, деятельность – это поведение, которое реализуется объектом, когда он находится в данном состоянии. Деятельность – это прерываемое действие: оно может выполняться до конца или быть прервано переходом в другое состояние.

Обозначается с помощью ключевого слова - метки деятельности do, которое специфицирует так называемую "ду-деятельность", выполняемую в течение всего времени, пока объект находится в данном *состоянии*, или до тех пор, пока не будет прервано внешним событием. При нормальном завершении внутренней деятельности генерируется соответствующее событие.

Пример. В нашем примере при оформлении заказа он должен быть оплачен (входное действие Оплатить заказ), обработка заказа подразумевает проверку оплаты и наличия товаров (деятельность Проверить оплату и наличие), переход в одно из состояний На комплектации, Укомплектован, Выдан означает смену статуса заказа (соответствует действию выхода Изменить статус) (рис. 114).

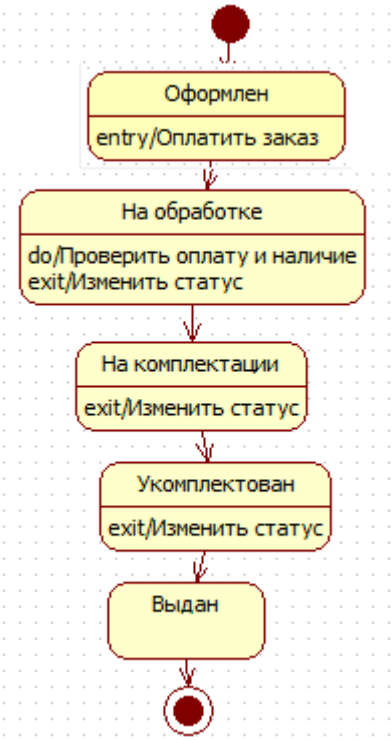


Рисунок 114. Диаграмма состояний с деятельностями

Изменение состояния объекта осуществляется с помощью переходов.

Переход (Transition) показывает, что объект, находящийся в некотором состоянии, должен выполнить некоторые действия и перейти в другое состояние, когда произойдет определенное событие, и будут выполнены соответствующие условия.

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.

Срабатывание <перехода> (fire) - выполнение перехода из одного состояния в другое.

На диаграмме переход изображается сплошной стрелкой. У перехода существует несколько спецификаций: событие, граничные условия, действия и посылаемые события.

<имя события>(<список параметров, разделенных запятыми>)[<сторожевое условие>]/<выражение действия>.

Событие (Event) – это то, что вызывает переход из одного состояния в другое. У события могут быть аргументы, которые записываются в скобках.

Граничные (ограждающие) условия определяют, когда может быть выполнен переход, а когда – нет. Условия записываются в квадратных скобках.

После условий может указываться ***действие*** - непрерываемое поведение, выполняемое как часть перехода.

Пример. В нашем примере если покупатель получил заказ, то это событие вызывает переход из состояния Укомплектован в состояние Выдан. Если же покупатель не получил свой заказ в течение двух недель, то заказ расформировывается, а деньги возвращаются покупателю на банковскую карту. Условие [Покупатель не забрал заказ в течение 2 недель] вызывает переход в состояние Расформирован при этом выполняется действие Вернуть деньги на карту. Окончательную диаграмму состояний можно видеть на рисунке 115.

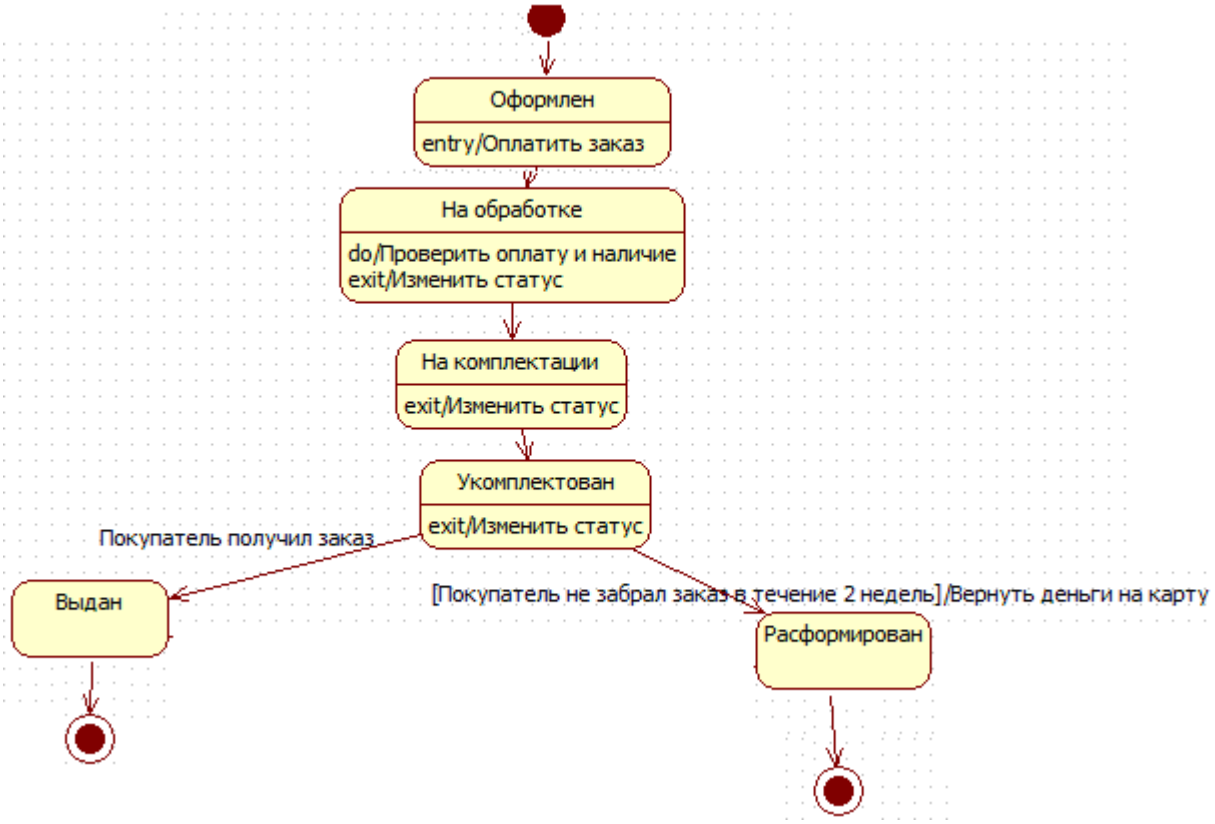


Рисунок 115. Окончательная диаграмма состояний объекта Заказ

Литература

1. Кватрани Т. Rational Rose 2000 и UML. Визуальное моделирование: Пер. с англ. – М.: ДМК Пресс, 2001 – 174 с. (http://www.knigafund.ru/books/106263)

2. Буч Г., Рамбо Д., Джекобсон А. Язык UML. Руководство пользователя. М.: ДМК Пресс, 2007 – 489 с. (http://www.knigafund.ru/books/106240)

3. Боггс У., Боггс М. UML и Rational Rose. М.: Лори, 2008 – 600 с.

4. Ипатова Э.Р., Ипатов Ю.В. Методологии и технологии системного проектирования информационных систем. – М.: Флинта, 2008. – 256с.

5. Фаулер М. UML. Основы. Краткое руководство по стандартному языку объектного моделирования. – СПб.: Символ-Плюс, 2011. – 192с.

6. Ларман К. Применение UML 2.0 и шаблонов проектирования. Введение в объектно-ориентированный анализ, проектирование и итеративную разработку. – М.: Вильямс, 2013. – 736с.

7. Избачков Ю., Петров В. и др. Информационные системы. – СПб.: Питер, 2010. – 544с.

8. Гвоздева Т.В., Баллод Б.А. Проектирование информационных систем. Серия «Высшее образование». – М.: Феникс, 2009. – 512 с.

9. Леоненков А. - Самоучитель UML – СПб.: БХВ-Петербург, 2007 – 576 с.

Интернет-источники

10. StarUML. The Open Source UML/MDA Platform. URL.: http://staruml.sourceforge.net/en/documentations.php (дата обращения 12.01.2013).

11. CASE // Википедия. Свободная энциклопедия. URL.: http://ru.wikipedia.org/wiki/CASE (дата обращения 23.03.2013).

12. Национальный открытый университет «Интуит» // Проектирование информационных систем, Введение в UML, Нотация и семантика языка UML. URL.: http://www.intuit.ru (дата обращения 12.12.12)

Приложение 1. Темы самостоятельных проектов

1. Проектирование системы интернет-бронирования гостиницы.
2. Проектирование системы реализации готовой продукции.
3. Проектирование системы интернет-заказов товаров магазина электроники.
4. Проектирование системы предоставления и запроса вакансий для бюро по трудоустройству.
5. Проектирование системы электронной записи клиентов нотариальной конторы.
6. Проектирование системы интернет-заказов у поставщиков автозапчастей.
7. Проектирование системы записи и учета прохождения курсов повышения квалификации.
8. Проектирование электронной системы учета оценок студентов
9. Проектирование электронной системы распределения нагрузки преподавателей.
10. Проектирование информационной системы страховой компании.
11. Проектирование системы контроля сроков и обслуживания клиентов ломбарда.
12. Проектирование электронной системы записи на прием пациентов частной клиники.
13. Проектирование системы учета кадров на предприятии.
14. Проектирование электронной системы заказа книг в библиотеке.
15. Проектирование театральной интернет-кассы.
16. Проектирование системы бронирования для проката автомобилей.
17. Проектирование системы учета рекламы в эфире телеканала.
18. Проектирование системы электронного расписания работы телеканала.
19. Проектирование системы интернет-заказов ювелирной мастерской.
20. Проектирование интернет-магазина одежды.
21. Проектирование электронной системы сдачи в аренду торговых площадей.
22. Проектирование системы продажи и бронирования билетов кинотеатра через интернет.
23. Проектирование интернет-афиши и справки кинотеатра.
24. Проектирование системы учета технического обслуживания станков.
25. Проектирование информационной системы турфирмы.
26. Проектирование системы покупки и бронирования билетов на поезд.
27. Проектирование информационной системы компании грузоперевозок.

28. Проектирование системы учета телефонных разговоров сотрудников.
29. Проектирование интернет-системы подачи заявок на оформление кредита.
30. Проектирование интернет-кабинета клиента банка.
31. Проектирование информационной системы агенства недвижимости.
32. Проектирование интернет-системы записи и учета скидок клиентов салона красоты.
33. Проектирование системы регистрации и контроля сообщений участников интернет-форума.
34. Проектирование системы доставки товаров из магазина.
35. Проектирование интернет-системы заказа и доставки пиццы.
36. Проектирование информационной системы детского сада.
37. Проектирование системы курсов дистанционного обучения.
38. Проектирование системы футбольных ставок.
39. Проектирование системы бронирования столиков и заказа блюд меню ресторана по интернету.
40. Проектирование системы обслуживания клиентов частной почтовой службы.
41. Проектирование системы учета сбыта продукции сельскохозяйственного предприятия.
42. Проектирование системы маркетинга предприятия.
43. Проектирование информационной системы компании прямых продаж косметики.
44. Проектирование каталога и системы заказов легковых автомобилей по интернету.
45. Проектирование системы гарантийного обслуживания электротоваров.

Приложение 2. Диаграммы проекта моделирования системы заказов магазина «Style»

Итак, в ходе работы над проектом моделирования системы заказов магазина «Style» были определены прецеденты, актеры, классы и объекты системы и отношения между ними, описан поток событий основного прецедента, а также построены диаграммы прецедентов (рис. П1 и П2), деятельности (рис. П7), классов (рис. П3, П4, П5), пакетов (рис. П6), взаимодействия (последовательности (рис. П8) и кооперации (рис. П9)), состояний (П10), которые и приводятся ниже.

Диаграммы прецедентов

Для системы заказов магазина «Style» мы определили актеров Покупатель, Сотрудник, Система Склад и прецеденты Заказ товаров, Управление статусом заказа, Получение информации о заказе.

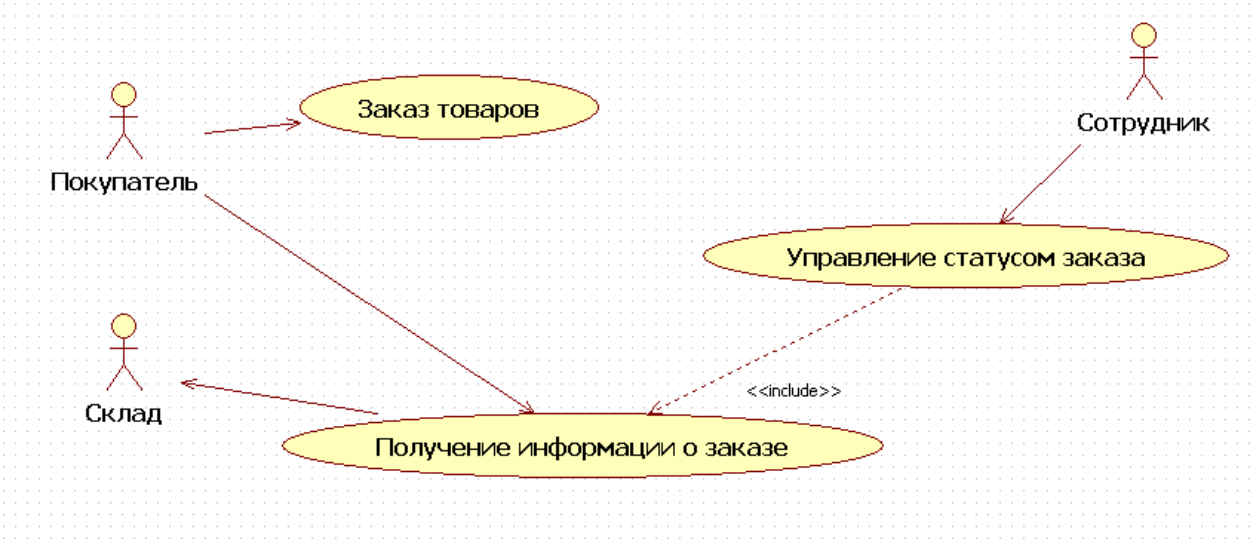


Рисунок П1. Основная диаграмма прецедентов

Для прецедента Заказ товаров мы построили дополнительную диаграмму вариантов использования (рис. П2).

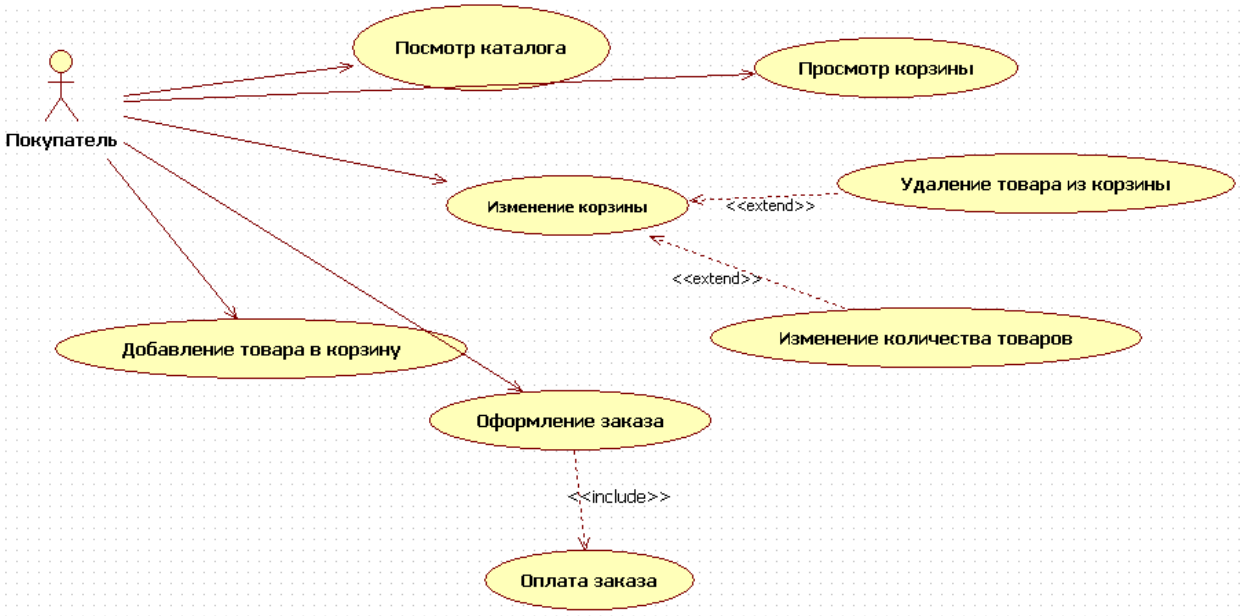


Рисунок П2. Дополнительная диаграмма прецедентов

Диаграммы классов прецедента Оформление заказа

Часто в проекте присутствует не одна диаграмма классов каждого сценария, а несколько: на одной изображают только классы (рис. П3), на других – классы с атрибутами и операциями (рис. П4) и отношениями (рис. П5).

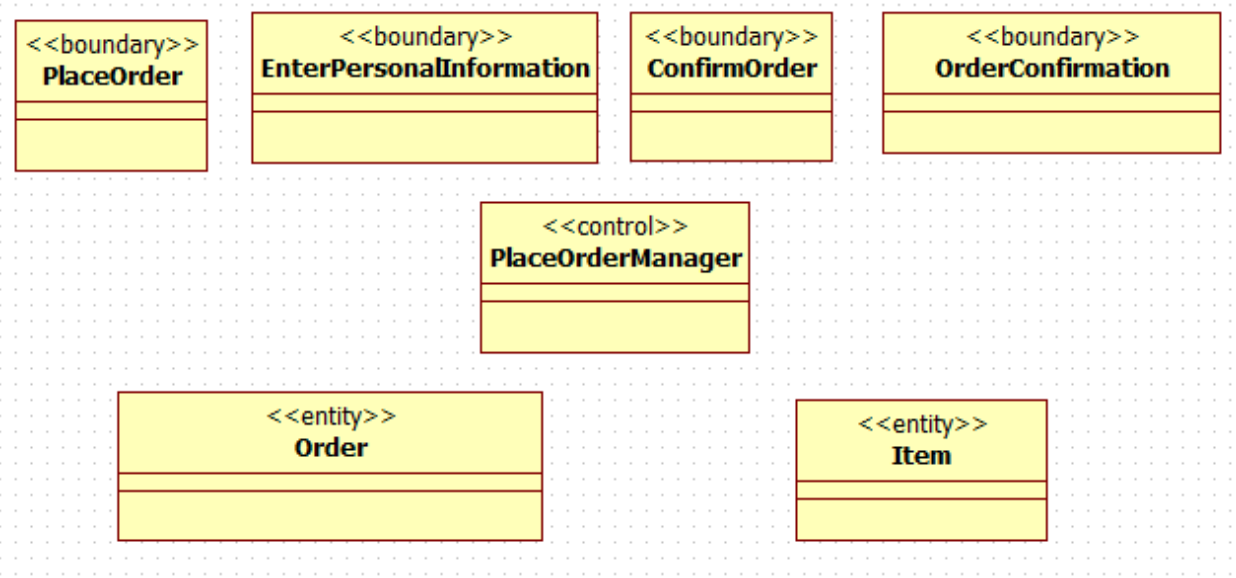


Рисунок П3. Диаграмма классов

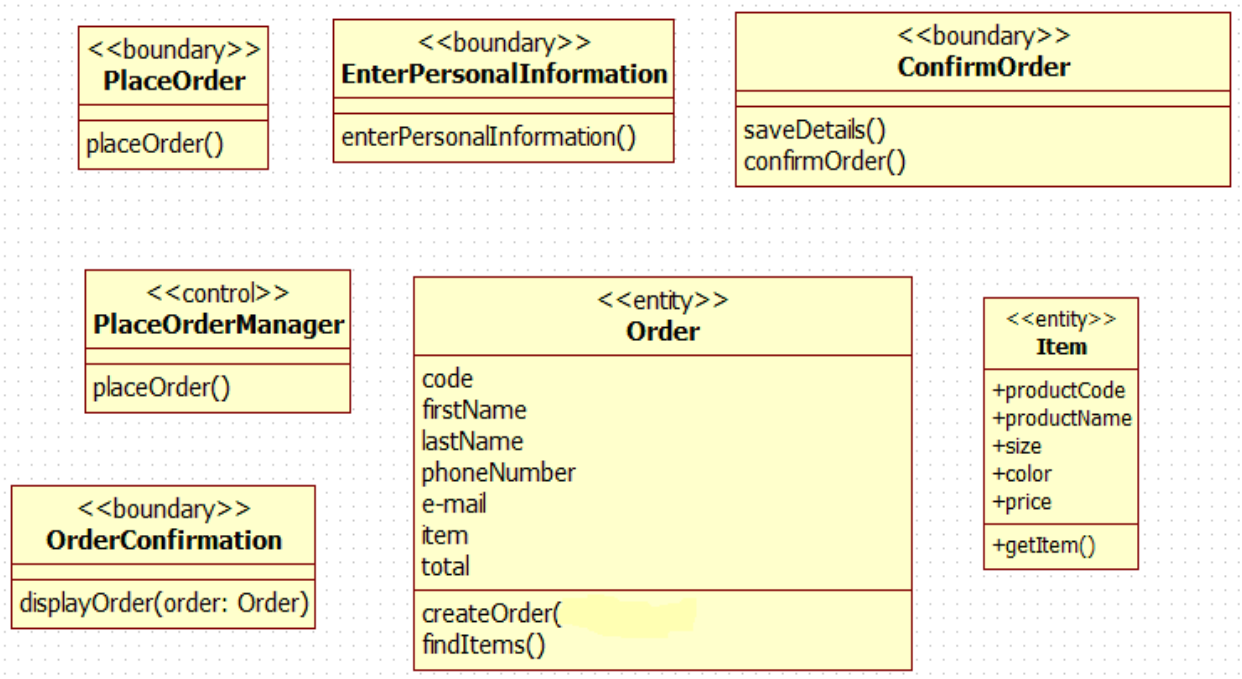


Рисунок П4. Диаграмма классов с атрибутами и операциями.

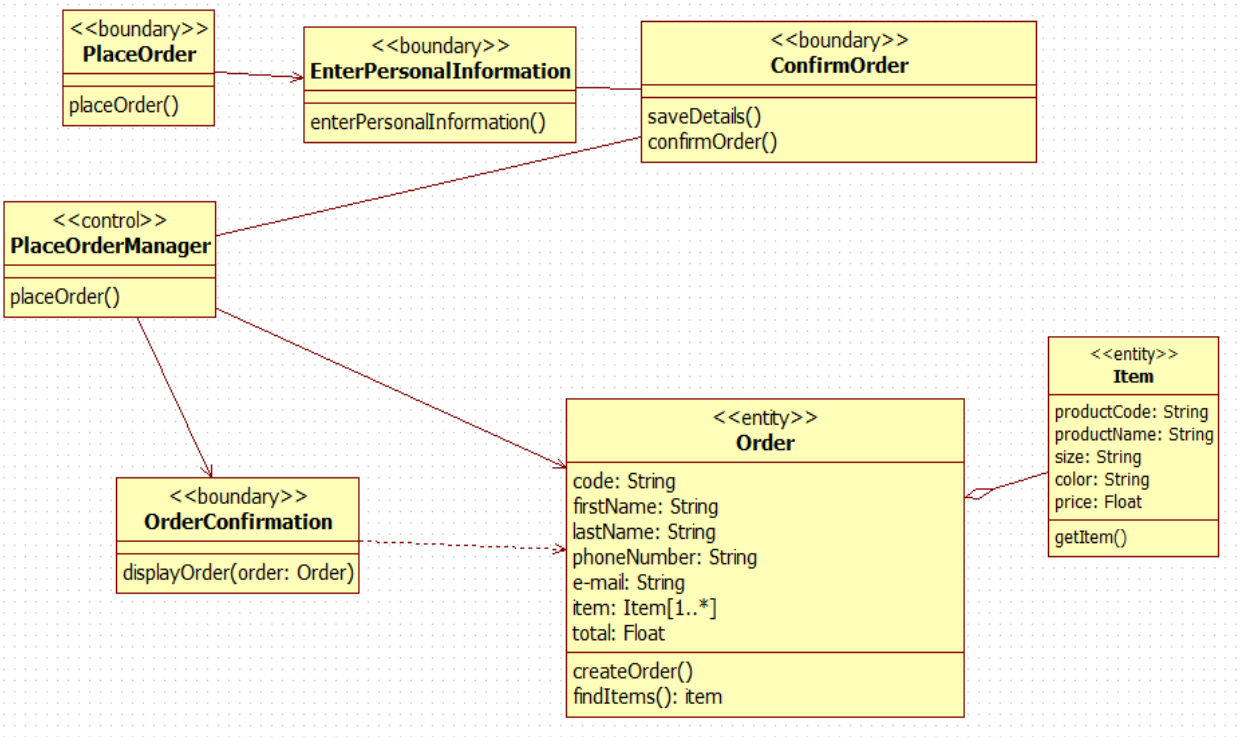


Рисунок П5. Диаграмма классов с отношениями

Диаграмма пакетов

Созданные ранее классы сценария Оформление заказа сгруппированы по пакетам: Граничные классы, Классы-сущности, Управляющие классы (рис. П6).

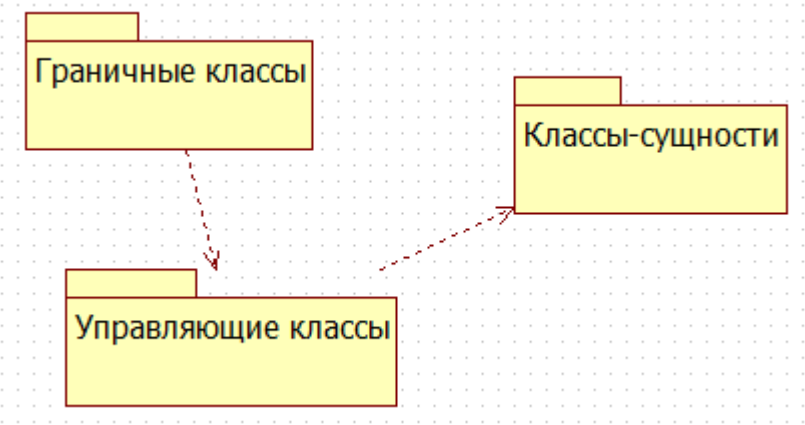


Рисунок П6. Диаграмма пакетов

Диаграмма деятельности прецедента Оформление заказа

Оформление заказа включает указание своих личных контактных данных, электронной почты и оплату заказа. Оформление начинается из корзины покупателя, когда он выбирает опцию «Оформить заказ».

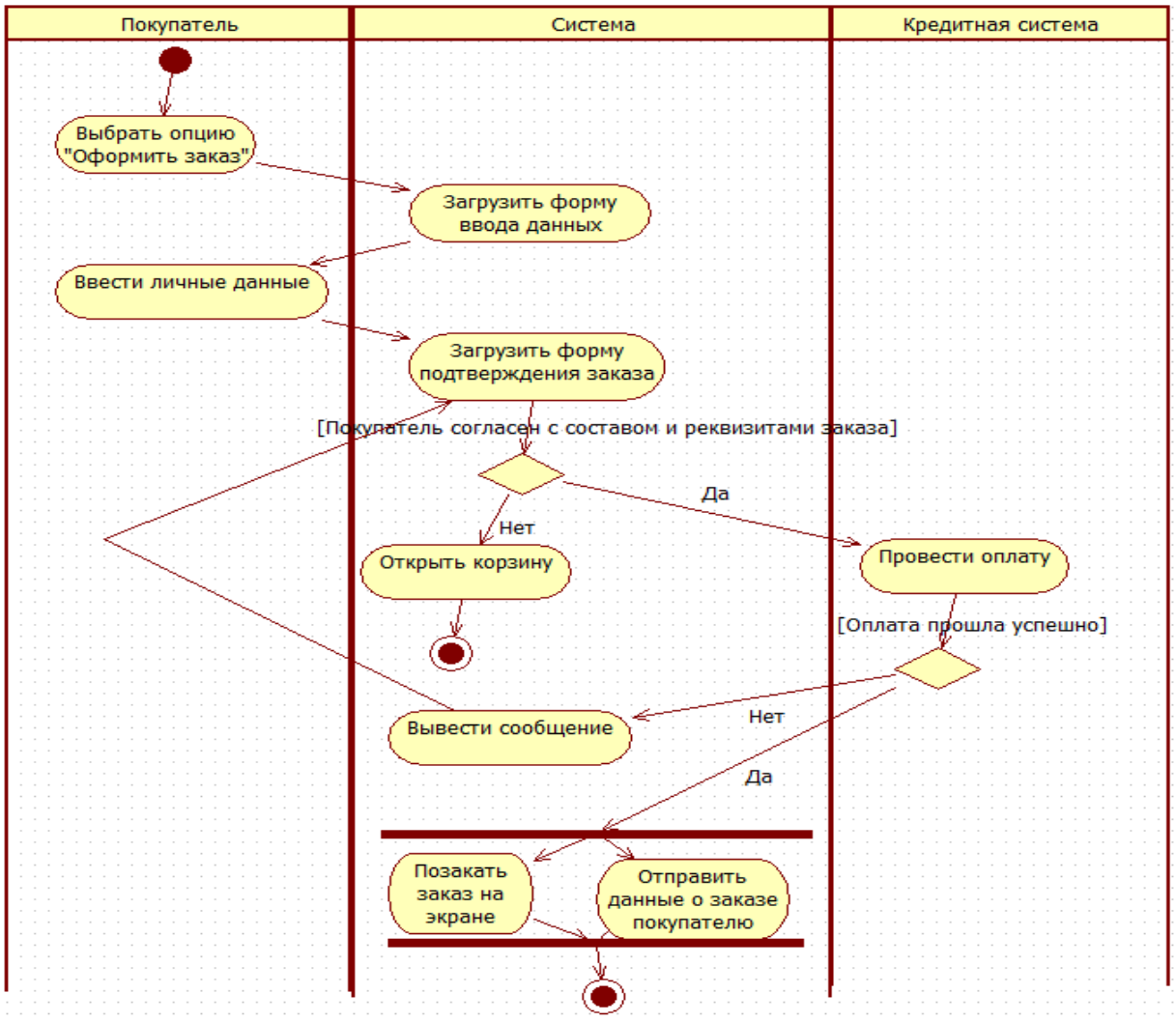


Рисунок П7. Диаграмма деятельности

Диаграмма последовательности сценария Оформление заказа

Покупатель выбирает опцию «Оформить заказ», затем открывается форма ввода личных данных покупателя и его кредитной карты (EnterPersonalInformation), на ней покупатель вводит свое имя, адрес, телефон, адрес электронной почты и кредитные данные. Информация принимается и открывается форма подтверждения заказа (ConfirmOrder). Фокус управления передается некоторому управляющему объекту (PlaceOrderManager), который обращается к внешней кредитной системе (Credit System) для проведения платежа. Если платеж прошел успешно, то PlaceOrderManager посылает сообщение объекту Заказ (Order), затем вызывает форму подтверждения заказа (OrderConfirmation). Объект Заказ (Order) обращается к объектам Товар (Item) для того, чтобы получить информацию о товарах и создает заказ.

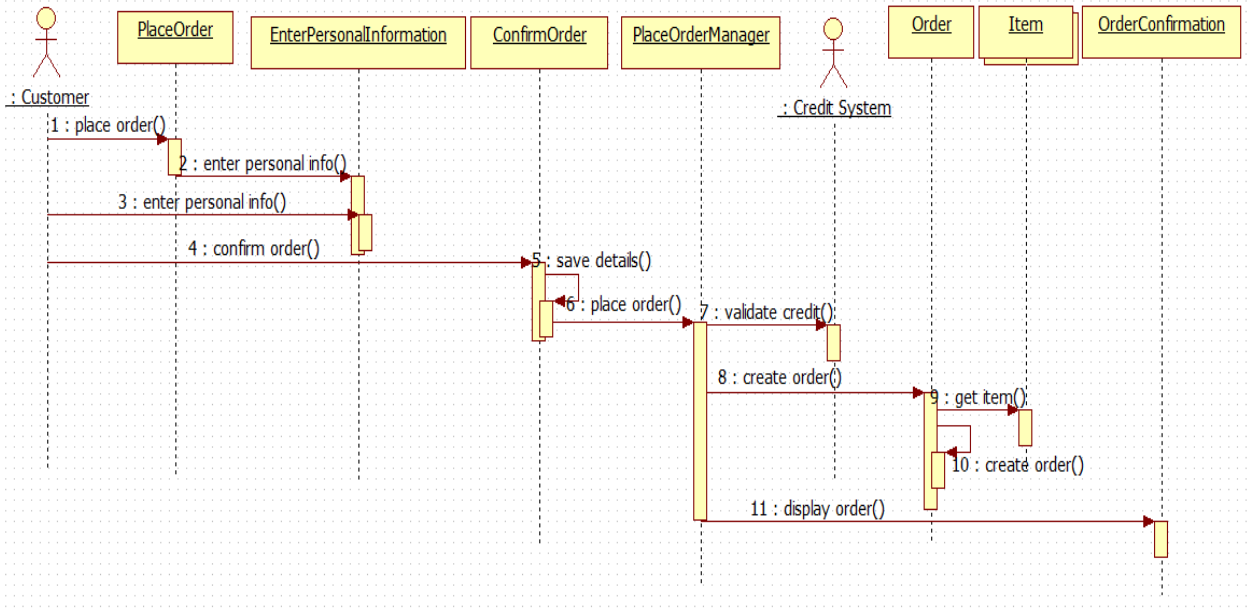


Рисунок П8. Диаграмма последовательности

Диаграмма кооперации сценария Оформление заказа

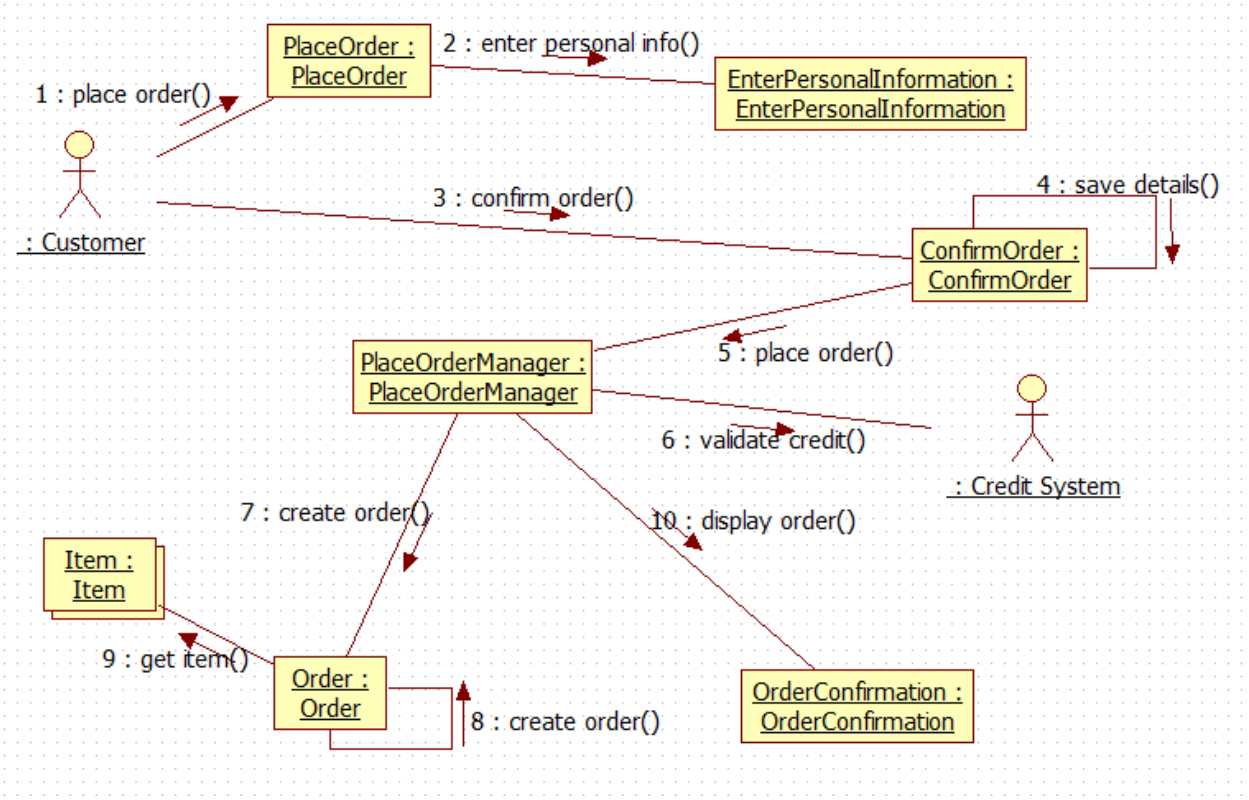


Рисунок П9. Диаграмма кооперации

Диаграмма состояний объекта Заказ

Заказ оформляется покупателем, затем находится На обработке в отделе продаж, после передается на комплектацию кладовщику, если покупатель получил заказ, то это событие вызывает переход из состояния Укомплектован в состояние Выдан. Если [Покупатель не забрал заказ в течение 2 недель], то заказ будет Расформирован, покупателю возвращаются его деньги.

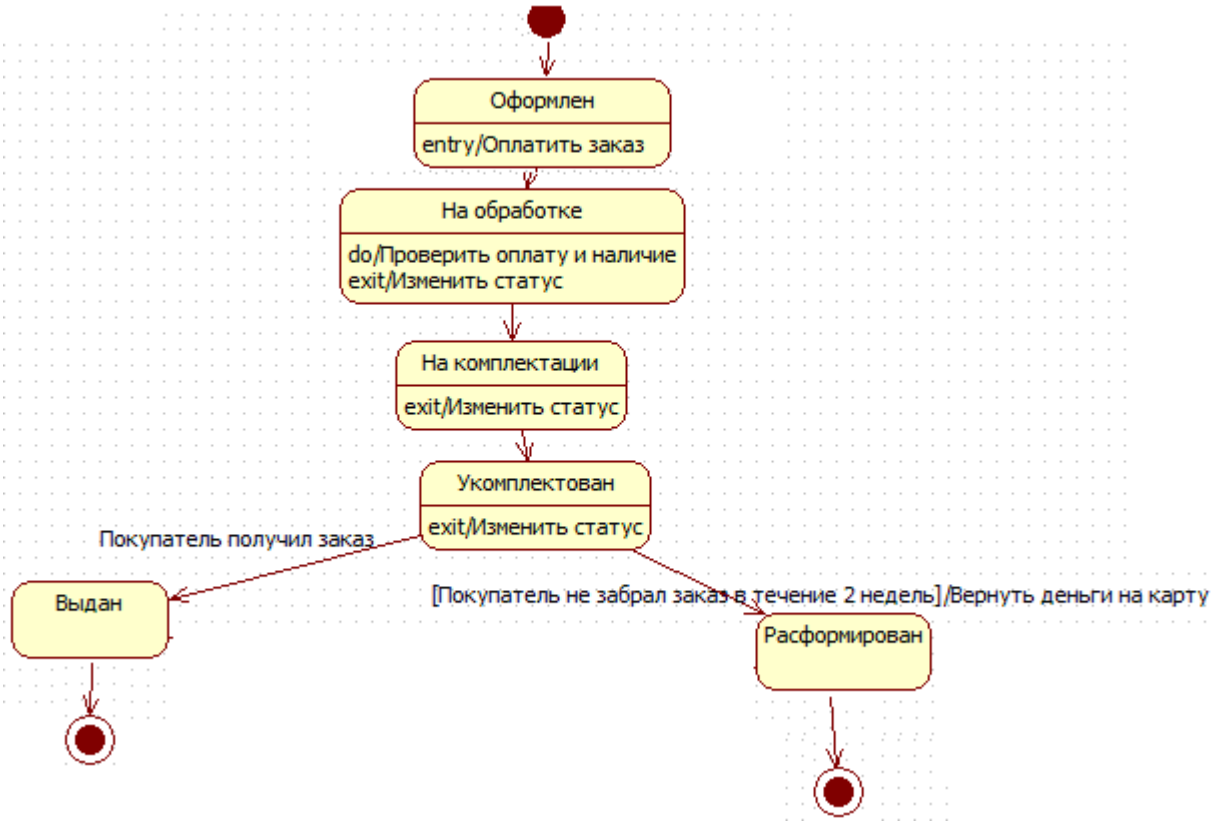


Рисунок П10. Диаграмма состояний

Предметный указатель