

Документ подписан простой электронной подписью
Информация о владельце:
ФИО: Пономарева Светлана Викторовна
Должность: Проректор по УР и НО
Дата подписания: 03.08.2022 23:09:38

СОДЕРЖАНИЕ

Практическая работа №12-13 Построение диаграмм состояний на языке UML с помощью MS Visio	4
Практическая работа №14-15 Построение диаграмм деятельности на языке UML с помощью MS Visio	14
Практическая работа №16-17 Построение диаграмм последовательностей на языке UML с помощью MS Visio	24
Практическая работа №18 Разработка диаграмм прецедентов с помощью Visual Paradigm for UML	29
Практическая работа №19 Разработка диаграмм классов с помощью Visual Paradigm for UML	37
Практическая работа №20 Разработка диаграмм состояний с помощью Visual Paradigm for UML	51
Практическая работа №21 Разработка диаграмм деятельности с помощью Visual Paradigm for UML	64
Практическая работа №22 Разработка диаграмм последовательности с помощью Visual Paradigm for UML	76

Практическая работа №12-13

Построение диаграмм состояний на языке UML с помощью MS Visio

Цель: изучение основ создания диаграмм состояний на языке UML, получение навыков построения диаграмм состояний, применение приобретенных навыков для построения объектно-ориентированных моделей определенной предметной области.

1 Задачи

Основными задачами практической работы являются:

- ознакомиться с теоретическими вопросами построения диаграмм состояний на языке UML;
- ознакомиться с теоретическими вопросами построения диаграмм состояний с помощью MS Visio.

2 Краткие теоретические сведения

Диаграмма состояний описывает процесс изменения состояний только одного класса, а точнее – одного экземпляра определенного класса, т. е. моделирует все возможные изменения в состоянии конкретного объекта. При этом изменение состояния объекта может быть вызвано внешними воздействиями со стороны других объектов или извне. Именно для описания реакции объекта на подобные внешние воздействия и используются диаграммы состояний.

Диаграмма состояний описывает возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла. Диаграмма состояний представляет динамическое поведение сущностей, на основе спецификации их реакции на восприятие некоторых конкретных событий.

Хотя диаграммы состояний чаще всего используются для описания поведения отдельных экземпляров классов (объектов), но они также могут быть применены для спецификации функциональности других компонентов моделей, таких как варианты использования, актеры, подсистемы, операции и методы.

Диаграмма состояний по существу является графом специального вида, который представляет некоторый автомат. Понятие автомата в контексте UML обладает довольно специфической семантикой, основанной на теории автоматов. Вершинами этого графа являются состояния и некоторые другие типы элементов автомата (псевдосостояния), которые изображаются соответствующими графическими символами. Дуги графа служат для обозначения переходов из состояния в состояние. Диаграммы состояний могут быть вложены друг в друга, образуя вложенные диаграммы более детального представления отдельных элементов модели.

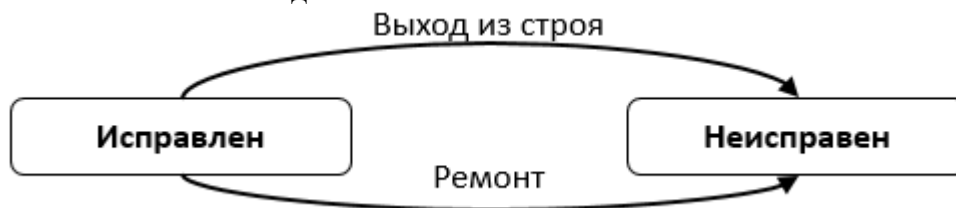


Рисунок 1 – Пример диаграммы состояний для технического устройства типа «Компьютер»

Состояние

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

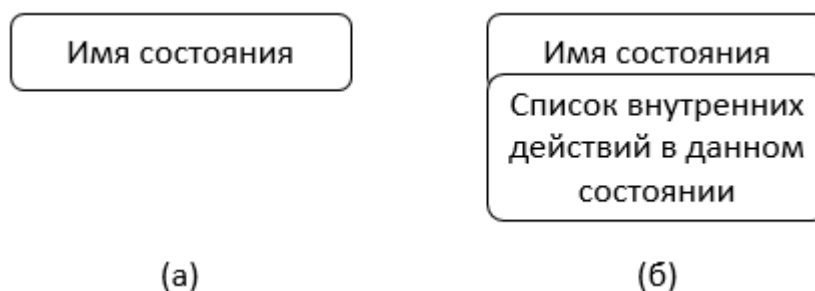


Рисунок 2 – Графическое обозначение состояния

Секция «Список внутренних действий» содержит перечень внутренних действий или деятельности, которые выполняются в процессе нахождения моделируемого элемента в данном состоянии. Каждое из действий записывается в виде отдельной строки и имеет следующий формат:

<метка-действия '/' выражение-действия>

Метка действия указывает на обстоятельства или условия, при которых будет выполняться деятельность, определенная выражением действия:

– **entry** – эта метка указывает на действие, специфицированное следующим за ней выражением действия, которое выполняется в момент входа в данное состояние (входное действие);

– **exit** – эта метка указывает на действие, специфицированное следующим за ней выражением действия, которое выполняется в момент выхода из данного состояния (выходное действие);

– **do** – эта метка специфицирует выполняющуюся деятельность («doactivity»), которая выполняется в течение всего времени, пока объект находится в данном состоянии, или до тех пор, пока не закончится вычисление, специфицированное следующим за ней выражением действия. В последнем случае при завершении события генерируется соответствующий результат;

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

Пример: Аутентификация входа

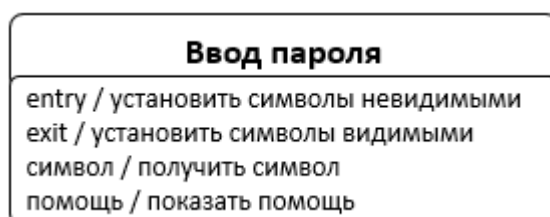


Рисунок 3 – Состояние для ввода пароля

Начальное и конечное состояния

Начальное состояние представляет собой частный случай состояния, которое не содержит никаких внутренних действий (псевдосостояния). В этом состоянии находится объект по умолчанию в начальный момент времени. Оно служит для указания на диаграмме состояний графической области, от которой начинается процесс изменения состояний.

Конечное (финальное) состояние представляет собой частный случай состояния, которое также не содержит никаких внутренних действий (псевдосостояния). В этом состоянии будет находиться объект по умолчанию после завершения работы автомата в конечный момент времени.



Рисунок 4 – Графическое изображение начального и конечного состояний

Переход

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

Переход осуществляется при наступлении некоторого события: окончания выполнения деятельности (doactivity), получении объектом сообщения или приемом сигнала. На переходе указывается имя события. Кроме того, на переходе могут указываться действия, производимые объектом в ответ на внешние события при переходе из одного состояния в другое. Срабатывание перехода может зависеть не

только от наступления некоторого события, но и от выполнения определенного условия, называемого сторожевым условием. Объект перейдет из одного состояния в другое в том случае, если произошло указанное событие и сторожевое условие приняло значение «истина».

На диаграмме состояний *переход изображается* сплошной линией со стрелкой, которая направлена в целевое состояние. Каждый переход может помечен строкой текста, которая имеет следующий общий формат:

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

Событие

Событие представляет собой спецификацию некоторого факта, имеющего место в пространстве и во времени. Про события говорят, что они «происходят», при этом отдельные события должны быть упорядочены во времени. После наступления некоторого события нельзя уже вернуться к предыдущим событиям, если такая возможность не предусмотрена явно в модели.

События играют роль стимулов, которые инициируют переходы из одних состояний в другие. В качестве событий можно рассматривать сигналы, вызовы, окончание фиксированных промежутков времени или моменты окончания выполнения определенных действий. Имя события идентифицирует каждый отдельный переход на диаграмме состояний и может содержать строку текста, начинающуюся со строчной буквы.

Сторожевое условие

Сторожевое условие (guardcondition), если оно есть, всегда записывается в прямых скобках после события и представляет собой некоторое булевское выражение.

Если сторожевое условие принимает значение «истина», то соответствующий переход может сработать, в результате чего объект перейдет в целевое состояние. Если же сторожевое условие принимает значение «ложь», то переход не может сработать, и при отсутствии других переходов объект не может перейти в целевое состояние по этому переходу. Однако вычисление истинности сторожевого условия происходит только после возникновения ассоциированного с ним события, инициирующего соответствующий переход.

В общем случае из одного состояния может быть несколько переходов с одним и тем же событием-триггером. При этом никакие два сторожевых условия не должны одновременно принимать значение «истина». Каждое из сторожевых условий необходимо вычислять всякий раз при наступлении соответствующего события.

Пример

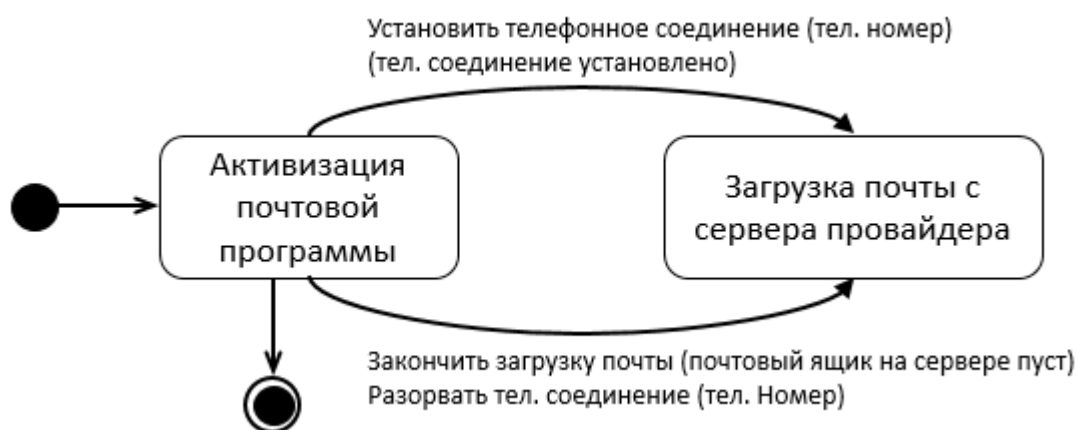


Рисунок 5 – Диаграмма состояний для моделирования почтовой программы клиента

Составное состояние и подсостояние

Составное состояние (compositestate) – такое сложное состояние, которое состоит из других вложенных в него состояний. Последние будут выступать по отношению к первому как подсостояния (substate).

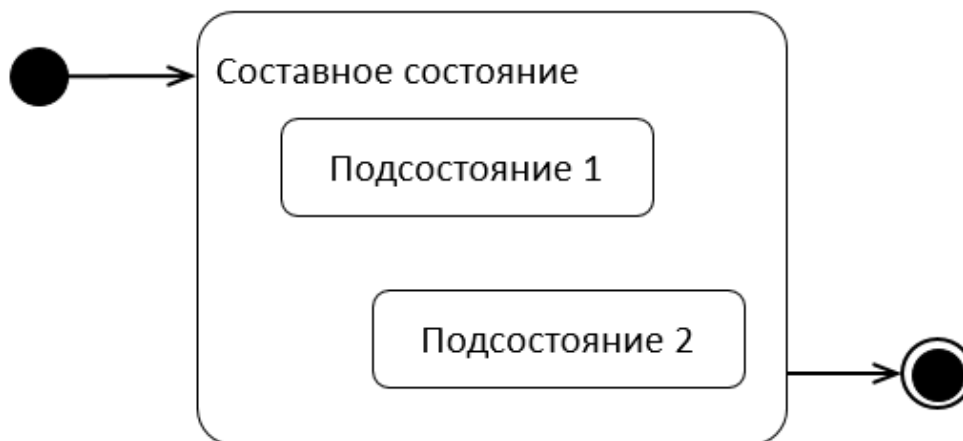


Рисунок 6 – Графическое изображение составного состояния

Последовательные подсостояния (sequential substates) используются для моделирования такого поведения объекта, во время которого в каждый момент времени объект может находиться в одном и только одном подсостоянии. Поведение объекта в этом случае представляет собой последовательную смену подсостояний, начиная от начального и заканчивая конечным подсостояниями. Хотя объект продолжает находиться в составном состоянии, введение в рассмотрение последовательных подсостояний позволяет учесть более тонкие логические аспекты его внутреннего поведения.

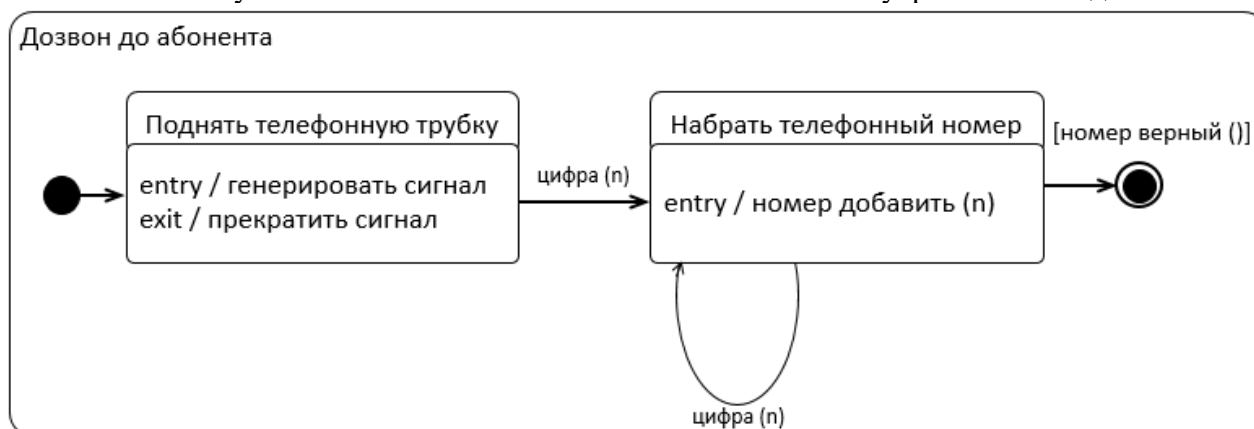


Рисунок 7 – Пример составного состояния с двумя вложенными последовательными подсостояниями

Параллельные подсостояния (concurrent substates) позволяют специфицировать два и более подавтомата, которые могут выполняться параллельно внутри составного события. Каждый из подавтоматов занимает некоторую область (регион) внутри составного состояния, которая отделяется от остальных горизонтальной пунктирной линией. Если на диаграмме состояний имеется составное состояние с вложенными параллельными подсостояниями, то объект может одновременно находиться в каждом из этих подсостояний.

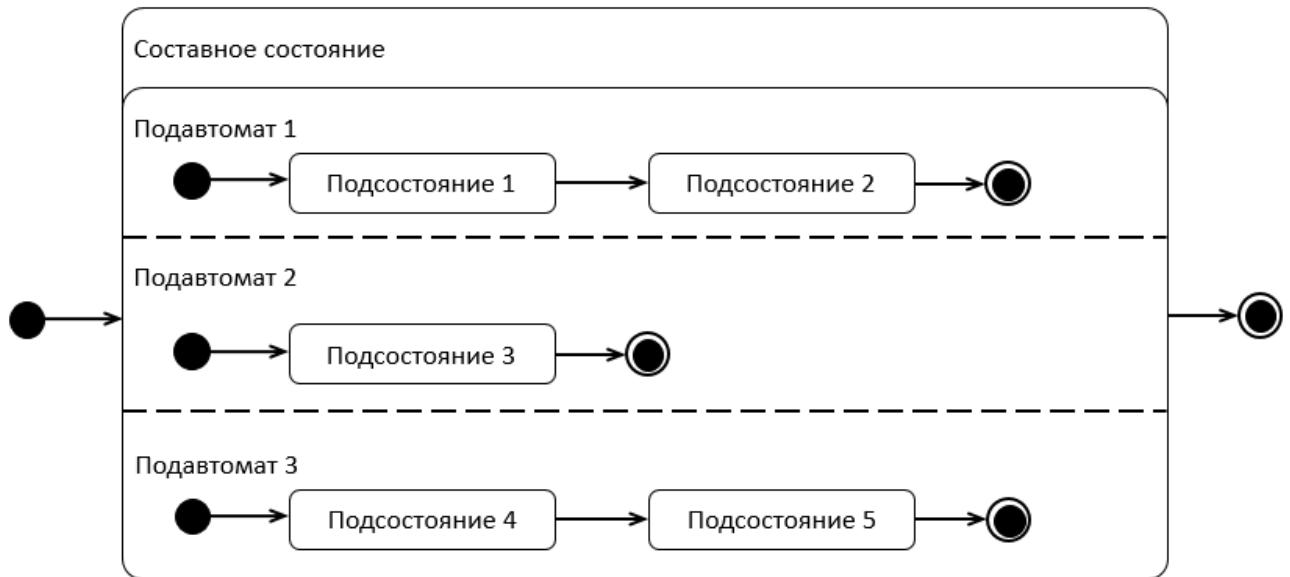


Рисунок 8 – Графическое изображение составного состояния с вложенными параллельными подсостояниями

Синхронизирующие состояния

Поведение параллельных подавтоматов независимо друг от друга, что позволяет реализовать многозадачность в программных системах. Однако в отдельных случаях может возникнуть необходимость учесть в модели синхронизацию наступления отдельных событий. Для этой цели в языке UML имеется специальное псевдосостояние, которое называется синхронизирующим состоянием.

Синхронизирующее состояние (synch state) обозначается небольшой окружностью, внутри которой помещен символ звездочки "*". Оно используется совместно с переходом-соединением или переходом-ветвлением для того, чтобы явно указать события в других подавтоматах, оказывающие непосредственное влияние на поведение данного подавтомата.

Для иллюстрации использования синхронизирующих состояний рассмотрим упрощенную ситуацию с моделированием процесса постройки дома. Предположим, что постройка дома включает в себя строительные работы (возведение фундамента и стен, возведение крыши и отделочные работы) и работы по электрификации дома (подведение электрической линии, прокладка скрытой электропроводки и установка осветительных ламп). Очевидно, два этих комплекса работ могут выполняться параллельно, однако между ними есть некоторая взаимосвязь.

В частности, прокладка скрытой электропроводки может начаться лишь после того, как будет завершено возведение фундамента и стен. А отделочные работы следует начать лишь после того, как будет закончена прокладка скрытой электропроводки. В противном случае отделочные работы придется проводить повторно. Рассмотренные особенности синхронизации этих параллельных процессов учтены на соответствующей диаграмме состояний с помощью двух синхронизирующих состояний.

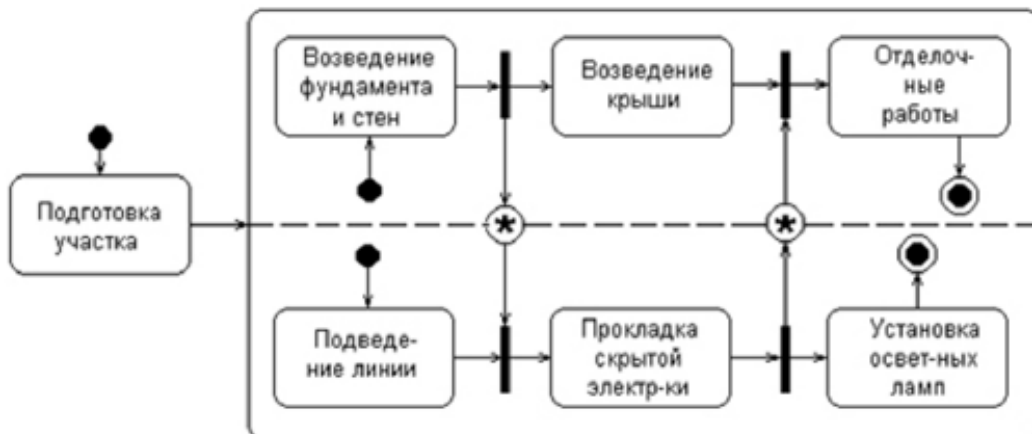


Рисунок 9 – Диаграмма состояний для примера со строительством дома

Примеры диаграмм состояний изображены на рисунках 10-12.

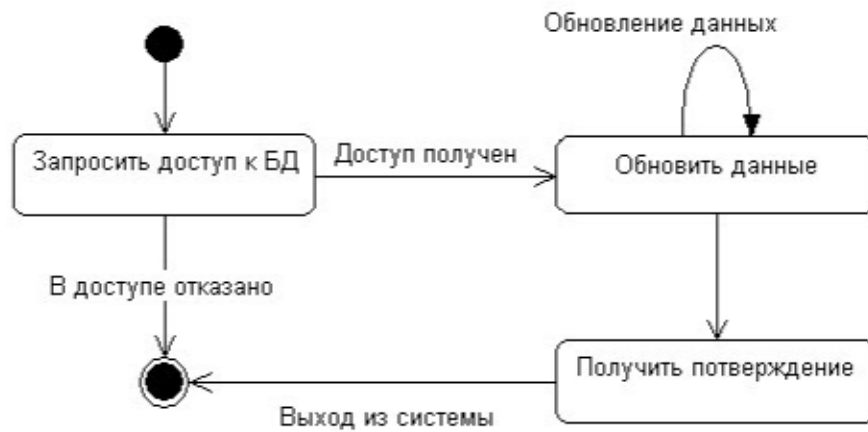


Рисунок 10 – Диаграмма состояний для моделирования запроса данных из БД



Рисунок 11 – Диаграмма состояний для моделирования поведения банкомата

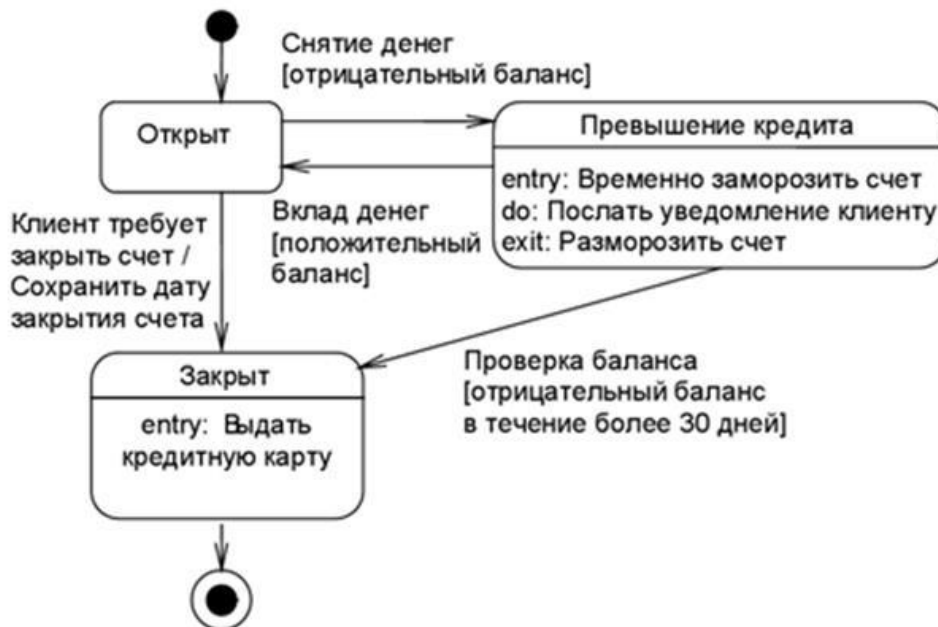


Рисунок 12 – Диаграмма состояний моделирования деятельности сотрудника банка

3 Методика выполнения

В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

1. Запустите MS Visio.
2. На экране выбора шаблона выберите категорию *Программы и БД* и в ней элемент *Схема модели UML*. Нажмите кнопку *Создать* в правой части экрана.
3. Окно программы примет вид, подобный рис. 13.

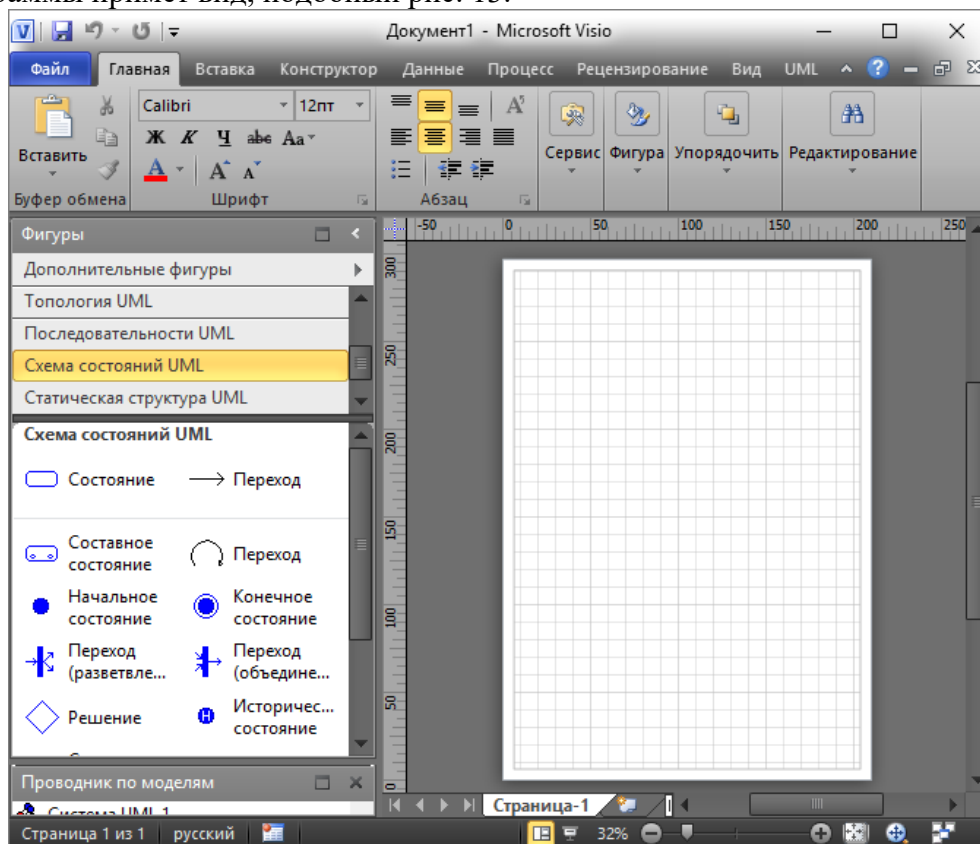


Рисунок 13 – Окно MS Visio для работы с диаграммами состояний

Клиент оформляет заказ. Класс Заказ имеет атрибут Статус. Проследим динамику движения заказов в системе с помощью *диаграммы состояний, составленной для класса Заказ*.

Данные о сформированном заказе поступают продавцу, который проверяет наличие товаров из заказа, проверяет оплату заказа, комплектует его и делает отметку о готовности. После оплаты заказа он выдается клиенту. Продавец делает отметку о том, что заказ выдан.

Если после проверки кредитного рейтинга клиента, он окажется отрицательным, то заказ будет отклонен.

Построим диаграмму состояний для класса Заказ. Для этого, в файле с диаграммой классов, созданной в практическом занятии 8, необходимо проделать следующие действия:

1. Щелкнуть правой кнопкой мыши по *классу Заказ*.
2. В контекстном меню выбрать пункт *Схемы*.
3. Т.к. в настоящее время уже созданных схем нет, нажать кнопку *Создать* и выбрать *Схема состояний*.
4. Переименовать созданный лист в *Схема состояний-Заказ*.
5. Построить диаграмму состояний для класса Заказ в соответствии с рисунком 14.

После формирования заказа он должен быть оплачен. Обработка заказа подразумевает проверку наличия товара и проверку оплаты. Переход в одно из состояний На комплектации, Укомплектован, Выдан означает смену Статуса заказа.

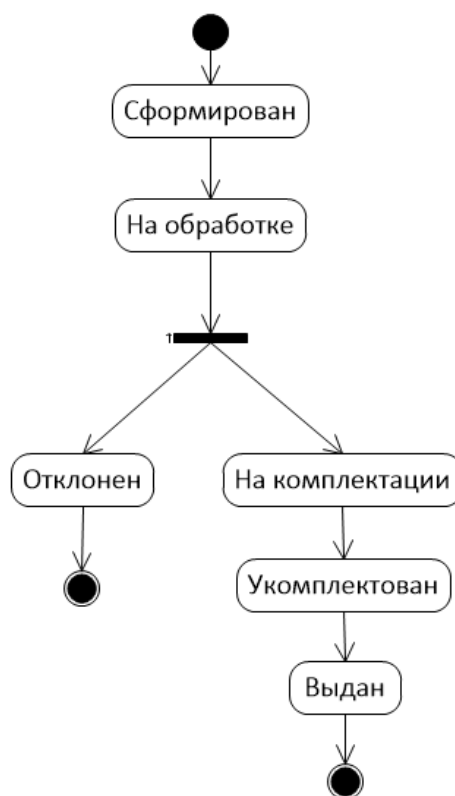


Рисунок 14 – Диаграмма состояний для класса Заказ

Далее опишем с помощью *диаграммы состояний процесс оплаты заказа клиентом*, которому соответствует класс ЗаказОплата.

Построим диаграмму состояний для проверки оплаты заказа.

Чтобы проверить оплату заказа, необходимо определить, существует ли сам заказ. Результатом проверки оплаты заказа является вывод либо сообщения о произведенной оплате с параметрами (дата оплаты), либо сообщения об ожидании оплаты.

Событием, предшествующим проверке оплаты заказа, является занесение информации о заказе в базу данных заказов.

Чтобы построить диаграмму состояний для класса ЗаказОплаты, необходимо проделать действия, описанные в пунктах 1-4 построения диаграммы состояний для класса Заказ. Полученная диаграмма должна иметь вид, изображенный на рис. 15.

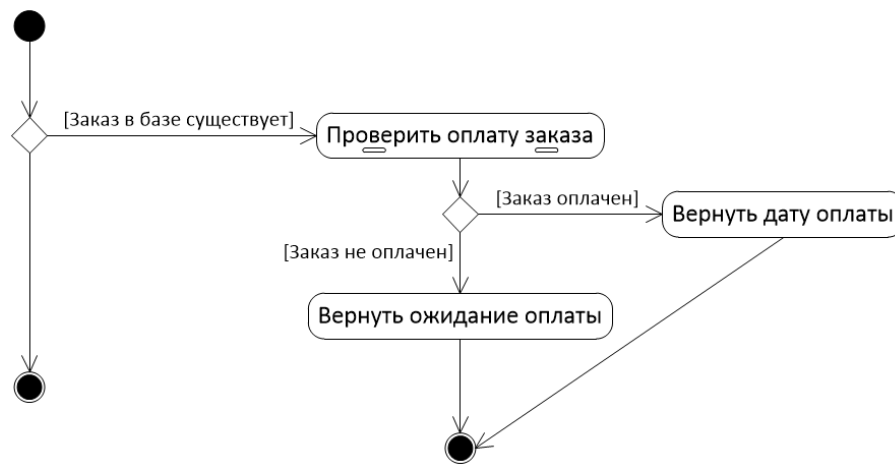


Рисунок 15 – Диаграмма состояний проверки платы заказа

На этой диаграмме есть составное состояние **«Проверить оплату заказа»**, т.к. оно включает в себя *проверку кредитного рейтинга клиента* и *проверку выбора варианта оплаты клиентом*.


Оплату заказа может произвести только клиент с положительным кредитным рейтингом, поэтому необходимым условием проверки оплаты заказа является проверка кредитоспособности клиента. Если клиент имеет отрицательный кредитный рейтинг, то заказ отклоняется, и на этом дальнейшие события не имеют смысла.

Если кредитный рейтинг клиента положительный, то необходимо проверить, выбрал ли клиент вариант оплаты. Событие, которое переводит систему в состояние ожидания выбора варианта оплаты клиентом, является получение сообщения о кредитоспособности клиента.

Оплата может быть произведена наличными средствами в магазине или с помощью безналичного расчета. В первом случае необходимо договориться с клиентом о дате и времени его прибытия в магазин. Во втором случае необходимо сообщить клиенту о наличии/поступлении товара. Событие, которое переводит систему в состояние ожидания оплаты, является выбор клиентом варианта оплаты.

Соответствующие диаграммы состояний имеют вид (рис. 16 и рис. 17).

Для создания диаграмм состояний, которые входят в состав составного состояния, нужно:

1. Щелкнуть правой кнопкой мыши по Составному состоянию и выбрать пункт Схема.
2. Либо, в Проводнике по моделям выделить название составного состояния и создать новую страницу с помощью кнопки .

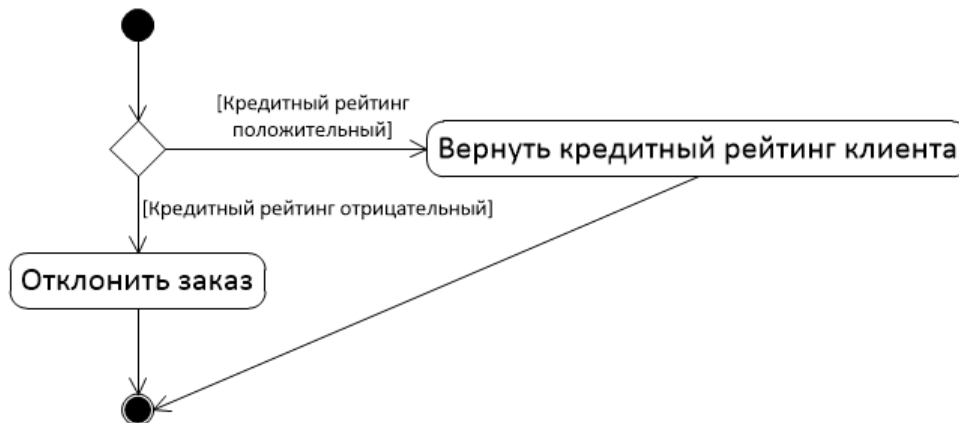


Рисунок 16 – Диаграмма состояний для проверки кредитного рейтинга клиента

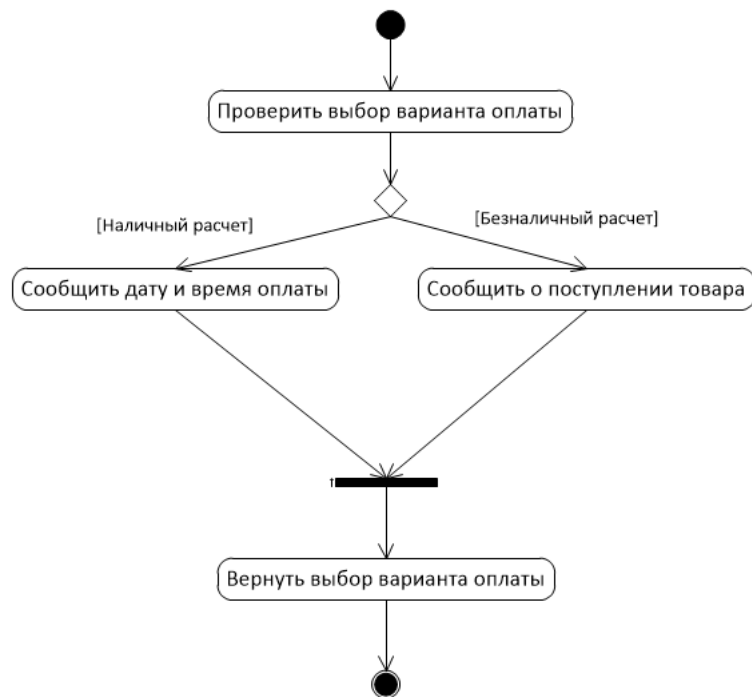


Рисунок 17 – Диаграмма состояний для проверки варианта оплаты

4 Задание

Построить диаграмму состояний в соответствии с вариантом.

Отчет по практическому занятию выполняется в формате MS Word, который содержит пошаговое описание процесса построения диаграммы, а также скриншоты результатов согласно заданию.

Создать диаграммы состояния не менее чем для трех классов, описанных в практическом занятии №10-11.

5 Варианты

1. «Отдел кадров»;
2. «Агентство аренды»;
3. «Аптека»;
4. «Ателье»;
5. «Аэропорт»;
6. «Библиотека»;
7. «Кинотеатр»;
8. «Поликлиника»;
9. «Автосалон»;
10. «Таксопарк».
11. «Издательство»;
12. «Прокат велосипедов»;
13. «Спортивный клуб».

6 Контрольные вопросы

1. Каково назначение диаграммы состояний?
2. Назовите основные элементы диаграммы состояний.
3. Как создать диаграмму состояний в VISIO?
4. В чем отличие диаграммы классов и состояний?

Практическая работа №14-15

Построение диаграмм деятельности на языке UML с помощью MS Visio

Цель: изучение основ создания диаграмм деятельности на языке UML, получение навыков построения диаграмм деятельности, применение приобретенных навыков для построения объектно-ориентированных моделей определенной предметной области.

1 Задачи

Основными задачами практической работы являются:

- ознакомиться с теоретическими вопросами построения диаграмм деятельности на языке UML;
- ознакомиться с теоретическими вопросами построения диаграмм деятельности с помощью MS Visio.

2 Краткие теоретические сведения

При моделировании поведения проектируемой или анализируемой системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и логической реализации выполняемых системой операций. Традиционно для этой цели использовались блок-схемы или структурные схемы алгоритмов. Каждая такая схема акцентирует внимание на последовательности выполнения определенных действий или элементарных операций, которые в совокупности приводят к получению желаемого результата.

Алгоритмические и логические операции, требующие выполнения в определенной последовательности, окружают нас постоянно. Например, чтобы позвонить по телефону, нам предварительно нужно снять трубку или включить его. Для приготовления кофе или заваривания чая необходимо вначале вскипятить воду. Чтобы выполнить ремонт двигателя автомобиля, требуется осуществить целый ряд нетривиальных операций, таких как разборка силового агрегата, снятие генератора и некоторых других.

С увеличением сложности системы строгое соблюдение последовательности выполняемых операций приобретает все более важное значение. Если попытаться заварить кофе холодной водой, то мы можем только испортить одну порцию напитка. Нарушение последовательности операций при ремонте двигателя может привести к его поломке или выходу из строя. Еще более катастрофические последствия могут произойти в случае отклонения от установленной последовательности действий при взлете или посадке авиалайнера, запуске ракеты, регламентных работах на АЭС.

Для моделирования процесса выполнения операций в языке UML используются так называемые **диаграммы деятельности**. Применяемая в них графическая нотация во многом похожа на нотацию диаграммы состояний, поскольку на диаграммах деятельности также присутствуют обозначения состояний и переходов. Отличие заключается в семантике состояний, которые используются для представления не деятельностей, а действий, и в отсутствии на переходах сигнатуры событий. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении этой операции в предыдущем состоянии. Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия, а дугами – переходы от одного состояния действия к другому.

Таким образом, диаграммы деятельности можно считать частным случаем диаграмм состояний. Основным направлением использования диаграмм деятельности является визуализация особенностей реализации операций классов, когда необходимо представить алгоритмы их выполнения. При этом каждое состояние может являться выполнением операции некоторого класса либо ее части, позволяя использовать диаграммы деятельности для описания реакций на внутренние события системы.

В контексте языка UML **деятельность (activity)** представляет собой некоторую совокупность отдельных вычислений, выполняемых автоматом. При этом отдельные элементарные вычисления могут приводить к некоторому результату или действию (action). На диаграмме деятельности отображается логика или последовательность перехода от одной деятельности к другой, при этом внимание фиксируется на результате деятельности. Сам же результат может привести к изменению состояния системы или возвращению некоторого значения.

Состояние действия и деятельности

Состояние деятельности (activity state) – состояние в графе деятельности, которое служит для представления процедурной последовательности действий, требующих определенного времени. Переход из состояния деятельности происходит после выполнения специфицированной в нем деятельности, при этом ключевое слово *do* в имени деятельности не указывается. Состояние деятельности не может иметь внутренних переходов, поскольку оно является элементарным.

Состояния деятельности могут быть подвергнуты дальнейшей декомпозиции, вследствие чего выполняемую деятельность можно представить с помощью других диаграмм деятельности. Состояния деятельности не являются атомарными, то есть могут быть прерваны. Предполагается, что для их завершения требуется заметное время.

Состояние деятельности можно представлять себе, как составное состояние, поток управления которого включает только другие состояния деятельности и действий.

Состояние действия (action state) является специальным случаем состояния с некоторым входным действием и, по крайней мере, одним выходящим из состояния переходом. Этот переход неявно предполагает, что входное действие уже завершилось. Состояние действия не может иметь внутренних переходов, поскольку оно является элементарным. Обычное использование состояния действия заключается в моделировании одного шага выполнения алгоритма (процедуры) или потока управления.

Графически состояние действия изображается фигурой, напоминающей прямоугольник, боковые стороны которого заменены выпуклыми дугами (рис. 1). Внутри этой фигуры записывается выражение действия (*action-expression*), которое должно быть уникальным в пределах одной диаграммы деятельности.



Рисунок 18 – Графическое обозначение состояния действия

Действие может быть записано на естественном языке, некотором псевдокоде или языке программирования. Никаких дополнительных или неявных ограничений при записи действий не накладывается. Рекомендуется в качестве имени простого действия использовать глагол с пояснительными словами (рис. 1а). Если же действие может быть представлено в некотором формальном виде, то целесообразно записать его на том языке программирования, на котором предполагается реализовывать конкретный проект (рис. 1б).

Иногда возникает необходимость представить на диаграмме деятельности некоторое сложное действие, которое, в свою очередь, состоит из нескольких более простых действий. В этом случае можно использовать специальное обозначение так называемого **состояния поддеятельности (subactivity state)**. Такое состояние является графом деятельности и обозначается специальной пиктограммой в правом нижнем углу символа состояния действия (рис. 2). Эта конструкция может применяться к любому элементу языка UML, который поддерживает «вложенность» своей структуры. При этом пиктограмма может быть дополнительно помечена типом вложенной структуры.

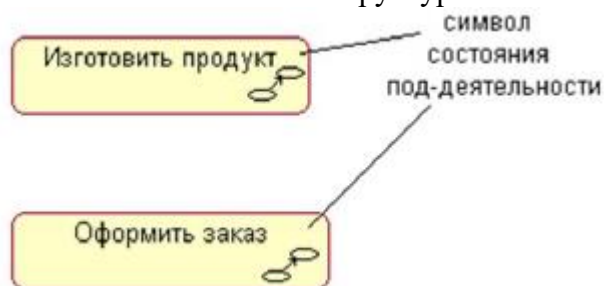


Рисунок 19 – Графическое обозначение состояния поддеятельности

Каждая диаграмма деятельности должна иметь единственное начальное и единственное конечное состояния. Они имеют такие же обозначения, как и на диаграмме состояний (см. практическое занятия №9). При этом каждая деятельность начинается в начальном состоянии и заканчивается в конечном состоянии. Саму диаграмму деятельности принято располагать таким образом, чтобы действия следовали сверху вниз. В этом случае начальное состояние будет изображаться в верхней части диаграммы, а конечное – в ее нижней части.

Переходы

При построении диаграммы деятельности используются только такие переходы, которые срабатывают сразу после завершения деятельности или выполнения соответствующего действия. Этот переход переводит деятельность в последующее состояние сразу, как только закончится действие в предыдущем состоянии. На диаграмме такой переход изображается сплошной линией со стрелкой.

Если из состояния действия выходит единственный переход, то он может быть никак не помечен. Если же таких переходов несколько, то сработать может только один из них. Именно в этом случае для каждого из таких переходов должно быть явно записано сторожевое условие в прямых скобках. При этом для всех выходящих из некоторого состояния переходов должно выполняться требование истинности только одного из них. Подобный случай встречается тогда, когда последовательно выполняемая деятельность должна разделиться на альтернативные ветви в зависимости от значения некоторого промежуточного результата. Такая ситуация получила название ветвления, а для ее обозначения применяется специальный символ.

Графически ветвление на диаграмме деятельности обозначается небольшим ромбом, внутри которого нет никакого текста (рис. 3). В этот ромб может входить только одна стрелка от того состояния действия, после выполнения которого поток управления должен быть продолжен по одной из взаимно исключающих ветвей. Принято входящую стрелку присоединять к верхней или левой вершине символа ветвления. Выходящих стрелок может быть две или более, но для каждой из них явно указывается соответствующее сторожевое условие в форме булевского выражения.

В качестве **примера** рассмотрим фрагмент известного алгоритма нахождения корней квадратного уравнения. В общем случае после приведения уравнения второй степени к каноническому виду: $a * x^2 + b * x + c = 0$ необходимо вычислить его дискриминант. Причем, в случае отрицательного дискриминанта уравнение не имеет решения на множестве действительных чисел, и дальнейшие вычисления должны быть прекращены. При неотрицательном дискриминанте уравнение имеет решение, корни которого могут быть получены на основе конкретной расчетной формулы.

Графически фрагмент процедуры вычисления корней квадратного уравнения может быть представлен в виде диаграммы деятельности с тремя состояниями действия и ветвлением (рис. 3). Каждый из переходов, выходящих из состояния «Вычислить дискриминант», имеет сторожевое условие, определяющее единственную ветвь, по которой может быть продолжен процесс вычисления корней в зависимости от знака дискриминанта. Очевидно, что в случае его отрицательности, мы сразу попадаем в конечное состояние, тем самым завершая выполнение алгоритма в целом.



Рисунок 20 – Фрагмент диаграммы деятельности для алгоритма нахождения корней квадратного уравнения

В рассмотренном примере, как видно из рис. 3, выполняемые действия соединяются в конечном состоянии. Однако это вовсе не является обязательным. Можно изобразить еще один символ ветвления, который будет иметь несколько входящих переходов и один выходящий.

Один из наиболее значимых недостатков обычных блок-схем или структурных схем алгоритмов связан с проблемой изображения параллельных ветвей отдельных вычислений. Поскольку распараллеливание вычислений существенно повышает общее быстродействие программных систем,

необходимы графические примитивы для представления параллельных процессов. В языке UML для этой цели используется специальный символ для разделения и слияния параллельных вычислений или потоков управления. Таким символом является **прямая черточка**.

Такая черточка изображается отрезком горизонтальной линии, толщина которой несколько шире основных сплошных линий диаграммы деятельности. При этом разделение (concurrent fork) имеет один входящий переход и несколько выходящих (рис. 4а). Слияние (concurrent join), наоборот, имеет несколько входящих переходов и один выходящий (рис. 4б).

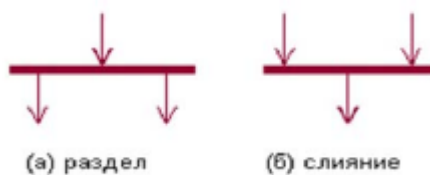


Рисунок 21 – Графическое обозначение разделения и слияния параллельных потоков управления

Для иллюстрации особенностей параллельных процессов выполнения действий рассмотрим **пример** с приготовлением напитка. Достоинство этого примера состоит в том, что он практически не требует никаких дополнительных пояснений в силу своей очевидности (рис. 5).

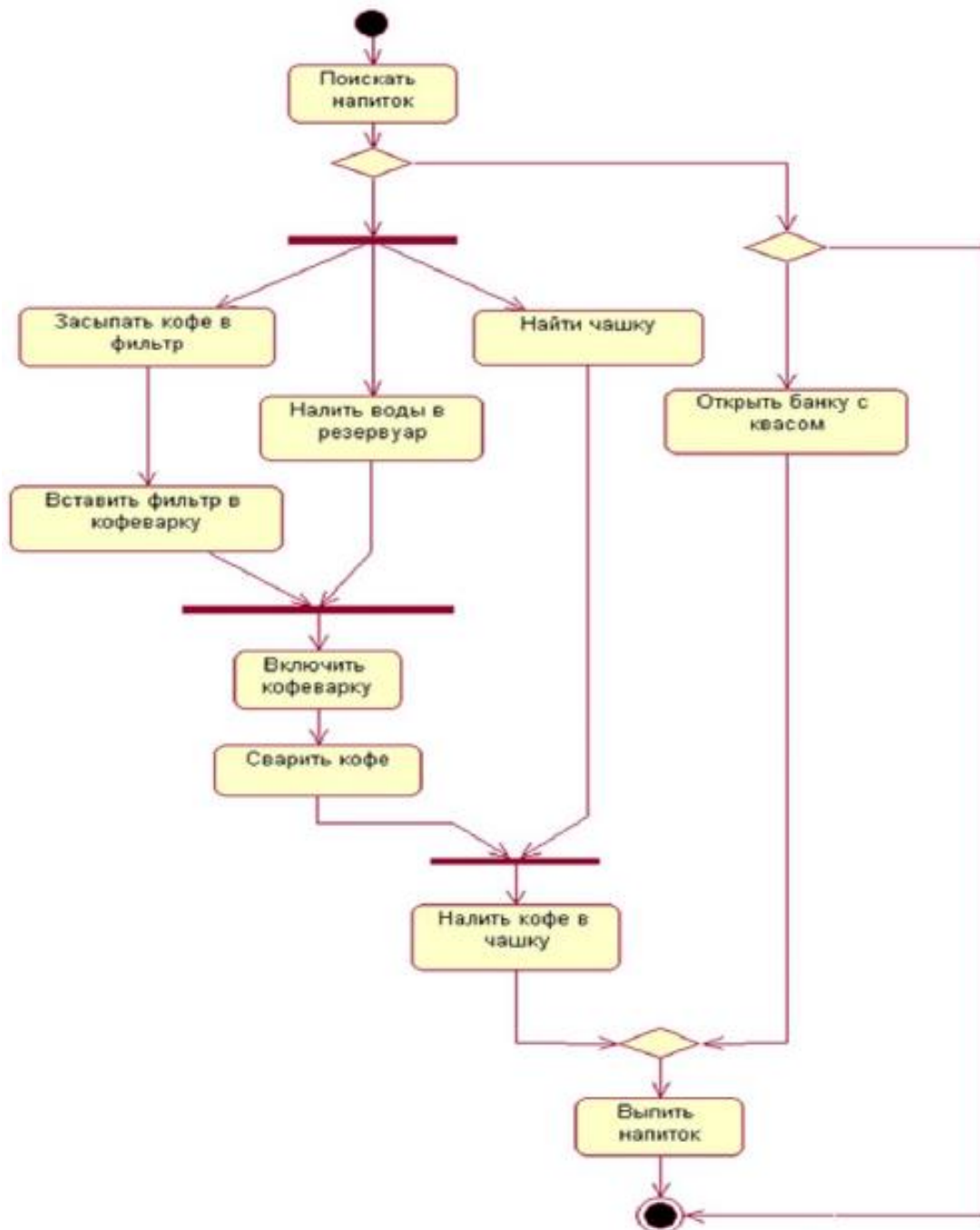


Рисунок 22 – Диаграмма деятельности для примера с приготовлением напитка

Хотя диаграмма деятельности предназначена для моделирования поведения систем, время в явном виде отсутствует на этой диаграмме. Ситуация здесь во многом аналогична диаграмме состояний.

Таким образом, диаграмма деятельности есть не что иное, как специальный случай диаграммы состояний, в которой все или большинство состояний являются действиями или состояниями поддеятельности. А все или большинство переходов являются переходами, которые срабатывают по завершении действий или под-деятельностей в состояниях источниках.

Дорожки

Диаграммы деятельности могут быть использованы не только для спецификации алгоритмов вычислений или потоков управления в программных системах. Не менее важная область их применения связана с моделированием бизнес-процессов. Действительно, деятельность любой компании (фирмы) также представляет собой не что иное, как совокупность отдельных действий, направленных на достижение требуемого результата. Однако применительно к бизнес-процессам желательно выполнение каждого действия ассоциировать с конкретным подразделением компании. В этом случае подразделение несет ответственность за реализацию отдельных действий, а сам бизнес-процесс представляется в виде переходов действий из одного подразделения к другому.

Для моделирования этих особенностей в языке UML используется специальная конструкция, получившее название *дорожки (swimlanes)*. Имеется в виду визуальная аналогия с плавательными дорожками в бассейне, если смотреть на соответствующую диаграмму. При этом все состояния действия на диаграмме деятельности делятся на отдельные группы, которые отделяются друг от друга вертикальными линиями. Две соседние линии и образуют дорожку, а группа состояний между этими линиями выполняется отдельным подразделением (отделом, группой, отделением, филиалом) компании (рис. 6).



Рисунок 23 – Вариант диаграммы деятельности с дорожками

Названия подразделений явно указываются в верхней части дорожки. Пересекать линию дорожки могут только переходы, которые в этом случае обозначают выход или вход потока управления в соответствующее подразделение компании. Порядок следования дорожек не несет какой-либо семантической информации и определяется соображениями удобства.

В качестве примера рассмотрим фрагмент диаграммы деятельности торговой компании, обслуживающей клиентов по телефону. Подразделениями компании являются отдел приема и оформления заказов, отдел продаж и склад.

Этим подразделениям будут соответствовать три дорожки на диаграмме деятельности, каждая из которых специфицирует зону ответственности подразделения. В данном случае диаграмма деятельности включает в себе не только информацию о последовательности выполнения рабочих действий, но и о том, какое из подразделений торговой компании должно выполнять то или иное действие (рис. 7).

Из указанной диаграммы деятельности сразу видно, что после принятия заказа от клиента отделом приема и оформления заказов осуществляется распараллеливание деятельности на два потока (переход-разделение). Первый из них остается в этом же отделе и связан с получением оплаты от клиента за заказанный товар. Второй инициирует выполнение действия по подбору товара в отделе продаж (модель товара, размеры, цвет, год выпуска и пр.). По окончании этой работы инициируется действие по отпуску товара со склада. Однако подготовка товара к отправке в торговом отделе начинается только после того, как будет получена оплата за товар от клиента и товар будет отпущен со склада (переход-соединение). Только после этого товар отправляется клиенту, переходя в его собственность.

Объекты

В общем случае действия на диаграмме деятельности выполняются над теми или иными объектами. Эти объекты либо инициируют выполнение действий, либо определяют некоторый результат этих действий. При этом действия специфицируют вызовы, которые передаются от одного объекта графа деятельности к другому. Поскольку в таком ракурсе объекты играют определенную роль в понимании процесса деятельности, иногда возникает необходимость явно указать их на диаграмме деятельности.

Для **графического представления объектов** используются прямоугольник класса, с тем отличием, что имя объекта подчеркивается. Далее после имени может указываться характеристика состояния объекта в прямых скобках. Такие прямоугольники объектов присоединяются к состояниям действия отношением зависимости пунктирной линией со стрелкой. Соответствующая зависимость определяет состояние конкретного объекта после выполнения предшествующего действия.

На диаграмме деятельности с дорожками расположение объекта может иметь некоторый дополнительный смысл. А именно, если объект расположен на границе двух дорожек, то это может означать, что переход к следующему состоянию действия в соседней дорожке ассоциирован с готовностью некоторого документа (объект в некотором состоянии). Если же объект целиком расположен внутри дорожки, то и состояние этого объекта целиком определяется действиями данной дорожки.

Возвращаясь к предыдущему примеру с торговой компанией, можно заметить, что центральным объектом процесса продажи является заказ или вернее состояние его выполнения. Вначале до звонка от клиента заказ как объект отсутствует и возникает лишь после такого звонка. Однако этот заказ еще не заполнен до конца, поскольку требуется еще подобрать конкретный товар в отделе продаж. После его подготовки он передается на склад, где вместе с отпуском товара заказ окончательно дооформляется. Наконец, после получения подтверждения об оплате товара эта информация заносится в заказ, и он считается выполненным и закрытым. Данная информация может быть представлена графически в виде модифицированного варианта диаграммы деятельности этой же торговой компании (рис. 8).

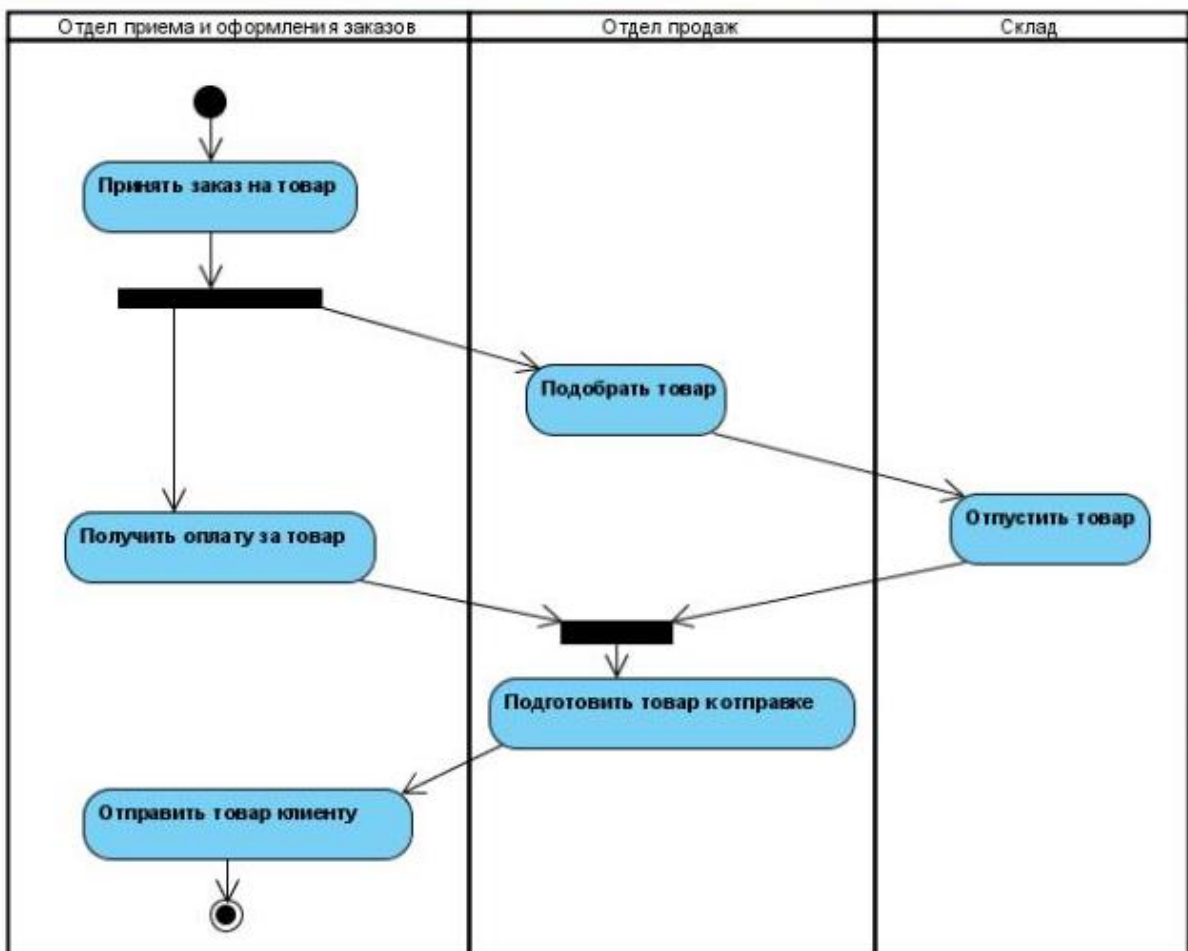


Рисунок 24 – Фрагмент диаграммы деятельности для торговой компании

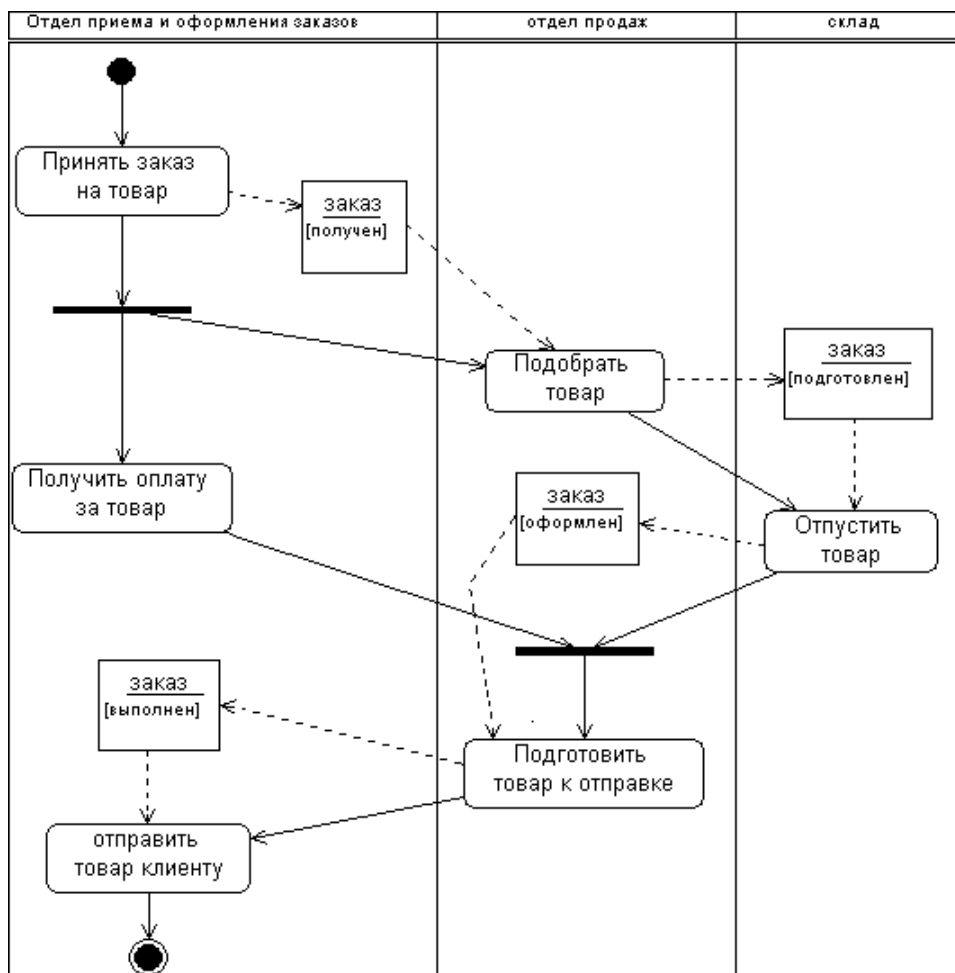


Рисунок 25 – Фрагмент диаграммы деятельности торговой компании с объектом-заказом

3 Методика выполнения

В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

1. Запустите MS Visio.
2. Откройте файл, созданный в практических занятиях №8-9 и содержащий диаграмму классов.
3. В проводнике по моделям должны отображаться все классы и диаграммы, созданные ранее (рис. 9).

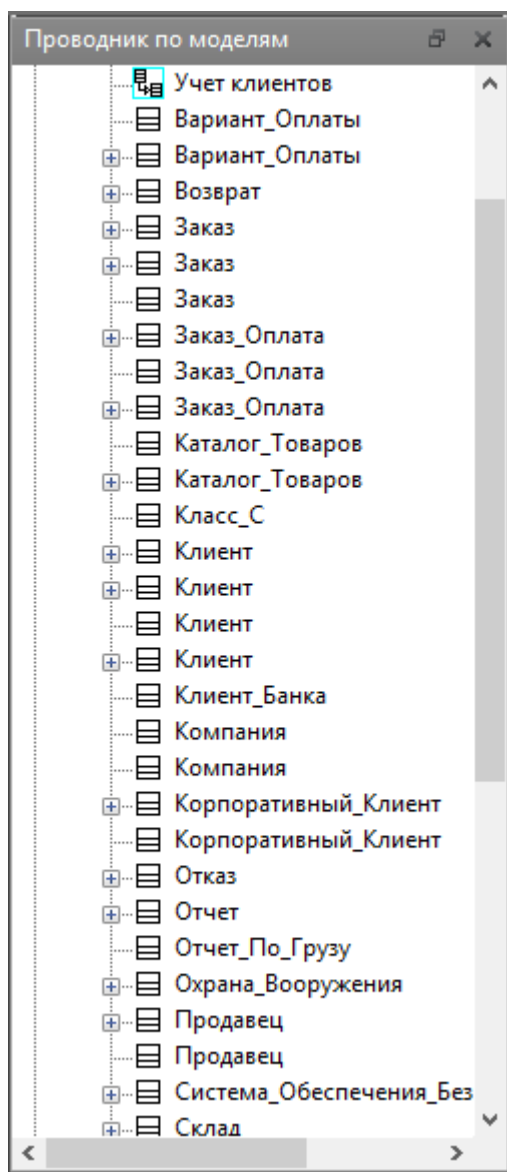


Рисунок 26 – Проводник по моделям

Опишем с помощью диаграммы деятельности процесс формирования заказа и выдачу товара. В бизнес-процессе участвуют 3 действующих лица: клиент, продавец и система оплаты. Следовательно, необходимо добавить 3 дорожки для распределения ответственности между этими лицами.

Для этого, в файле с диаграммой классов, созданной в практическом занятии 8, необходимо проделать следующие действия:

6. Щелкнуть правой кнопкой мыши по классу *Заказ*.
7. В контекстном меню выбрать пункт *Схемы*.
8. Нажать кнопку *Создать* и выбрать *Деятельность*.
9. Переименовать созданный лист в *Деятельность-Заказ*.
10. Построить диаграмму деятельности для класса *Заказ*. Для это выполните действия, описанные ниже.
 - а. Добавить 3 элемента *Дорожка* и изменить их названия на *Клиент*, *Продавец* и *Система оплаты* соответственно.
 - б. Добавить элементы *Начальное состояние*, *Конечное состояние*, *Состояние действия*, *Решение*, *Переход (объединение)*, изменить их названия и задать расположение в соответствии с рисунком 10.

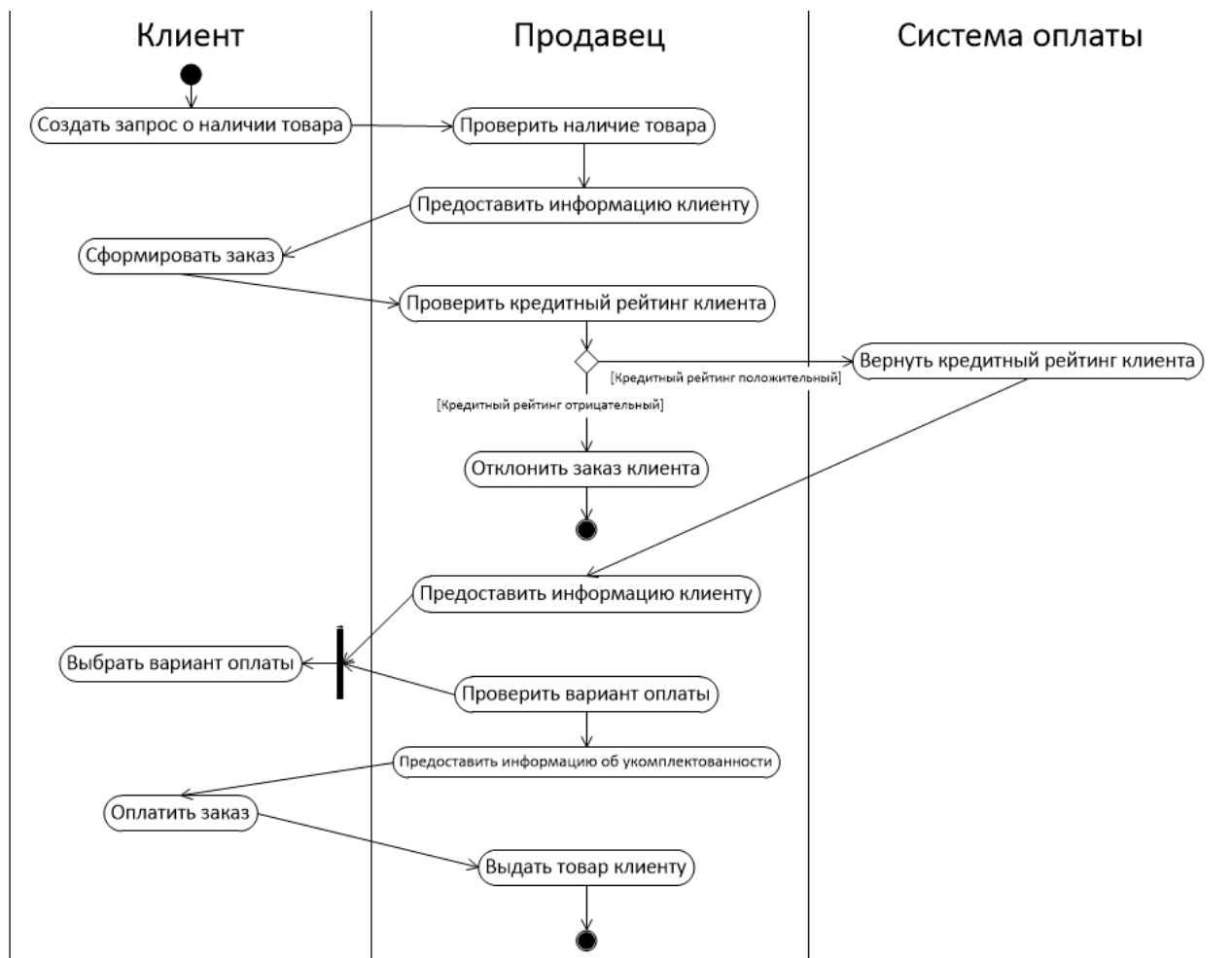


Рисунок 27 – Диаграмма деятельности

4 Задание

Построить диаграмму деятельности в соответствии с вариантом.

Отчет по практическому занятию выполняется в формате MS Word, который содержит пошаговое описание процесса построения диаграммы, а также скриншоты результатов согласно заданию.

Создать диаграммы деятельности не менее чем для трех классов, описанных в практическом занятии №10-11.

5 Варианты

1. «Отдел кадров»;
2. «Агентство аренды»;
3. «Аптека»;
4. «Ателье»;
5. «Аэропорт»;
6. «Библиотека»;
7. «Кинотеатр»;
8. «Поликлиника»;
9. «Автосалон»;
10. «Таксопарк».
11. «Издательство»;
12. «Прокат велосипедов»;
13. «Спортивный клуб».

6 Контрольные вопросы

1. Дайте определение понятию «диаграмма деятельности».
2. Опишите назначение диаграммы деятельности.

3. Дайте определение понятиям «состояние деятельности» и «состояние действия». Графическое изображение состояния.

4. Приведите пример ветвления и параллельных потоков управления процессами на диаграмме деятельности.

5. Какие переходы используются на диаграмме деятельности?

6. Что представляет собой дорожка на диаграмме деятельности?

7. Как графически изображаются объекты на диаграмме деятельности?

Практическая работа №16-17

Построение диаграмм последовательностей на языке UML с помощью MS Visio

Цель: изучение основ создания диаграмм последовательностей на языке UML, получение навыков построения диаграмм последовательностей, применение приобретенных навыков для построения объектно-ориентированных моделей определенной предметной области.

1 Задачи

Основными задачами практической работы являются:

- ознакомиться с теоретическими вопросами построения диаграмм последовательностей на языке UML;
- ознакомиться с теоретическими вопросами построения диаграмм последовательностей с помощью MS Visio.

2 Краткие теоретические сведения

Диаграммы последовательностей описывают взаимодействия множества объектов, включая сообщения, которыми они обмениваются.

В отличие от диаграммы классов, на которой изображаются абстрактные элементы в виде классов, на диаграмме последовательностей используются конкретные экземпляры классов – **объекты**. Объекты отображаются прямоугольником без полей. Для того чтобы подчеркнуть, что это экземпляр абстрактной сущности, название объекта подчеркивается. При необходимости через двоеточие после названия можно указать сущность (класс) экземпляром которой является этот объект. Отметим, что объект может быть экземпляром не только класса, но и других абстракций, например, актера. Обратите внимание, что при указании в качестве классификатора актера изменится графическое обозначение объекта (рис. 1).

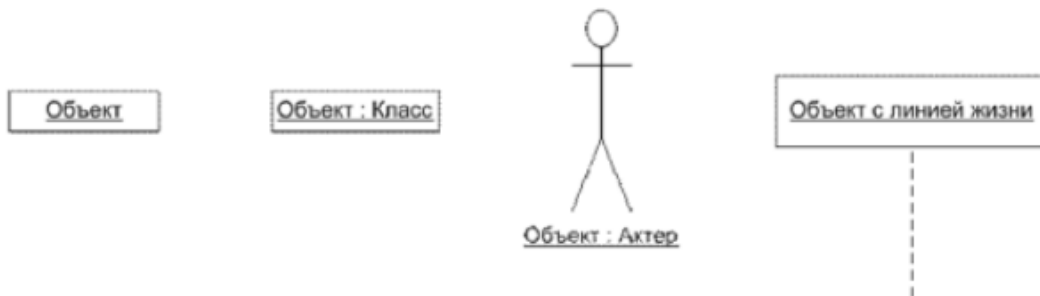


Рисунок 28 – Графическое обозначение объекта UML

На диаграмме последовательностей у объекта может присутствовать **линия жизни**, на которой отмечаются происходящие с объектом события. Линия жизни отображается пунктирной линией (рис. 2).

Между собой объекты могут быть связаны связями. **Связь** – это экземпляр отношения ассоциация, и имеет такое же графическое обозначение, что и ассоциация.

На диаграмме последовательностей объекты обмениваются сообщениями. **Сообщение** – это спецификация передачи данных от одного объекта другому, который предполагает какое-то ответное действие. Графически сообщение обозначается сплошной линией со стрелкой.

Часто операция вызывает какую-либо операцию в объекте. Очевидно, что класс, экземпляром которого является объект, должен иметь такую операцию. Привязка сообщения к операции класса объекта выполняется в свойствах сообщения (рис. 2).

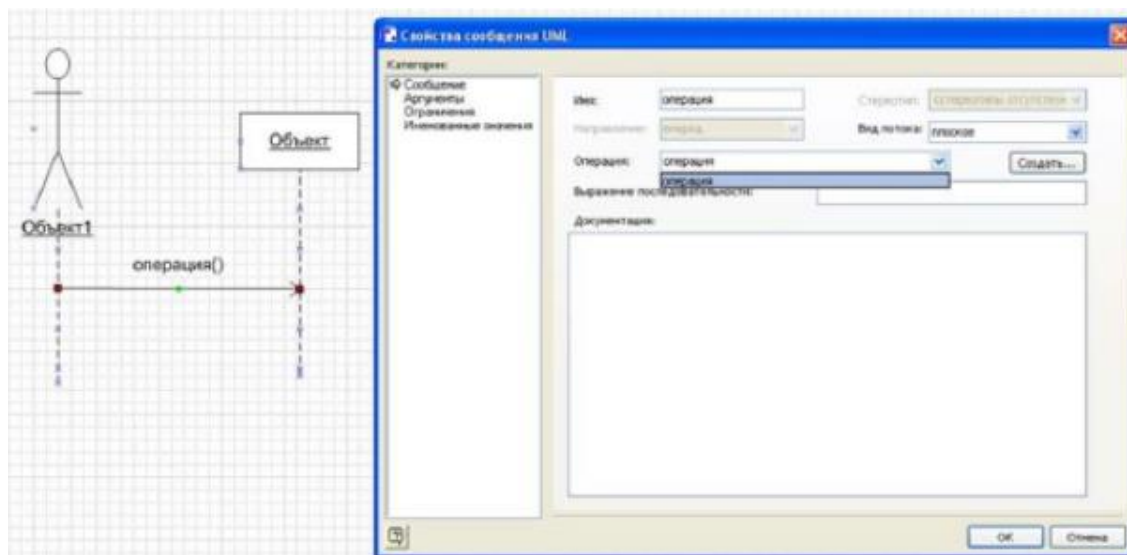


Рисунок 29 – Сообщение UML и его свойства

При построении диаграммы классов обычно определяются только основные свойства сущностей, а такие детали, как операции, удобно создавать при построении диаграммы последовательности, для чего в свойствах сообщения UML есть кнопка создания операции.

Диаграммы последовательностей, как и другие диаграммы для отображения динамических свойств системы, могут быть выполнены в контексте многих сущностей UML. Они могут описывать поведение системы в целом, подсистемы, класса или операции класса и др. К сожалению, Visio недостаточно гибка в плане поддержки раскрытия содержания отдельных элементов с помощью других диаграмм. Например, кликнув правой кнопкой мыши по классу можно обнаружить, что для его описания можно создать лишь диаграммы классов, состояний и деятельности. Поэтому возможность привязать диаграмму последовательностей к элементу, который она реализует, средствами Visio невозможно, эту связь нужно подразумевать.

Диаграммы последовательностей будем делать в контексте прецедентов с диаграммы прецедентов, реализуя те функции, которые должна выполнять наша система.

При построении динамических диаграмм используется уже разработанная структура информационной системы. Для диаграммы последовательностей не нужно придумывать объекты, а достаточно определить, экземпляры каких классов участвуют в этом действии.

Определив необходимые объекты (как экземпляры классов, так и экземпляры актеров), вторым этапом построения диаграмм последовательностей определяются сообщения, пересылаемые между актерами. Фактически определяется последовательность шагов, для выполнения нужного действия.

3 Методика выполнения

В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

1. Запустите MS Visio.
2. Откройте файл, созданный в практических занятиях №8-10 и содержащий диаграмму классов.
3. В проводнике по моделям должны отображаться все классы и диаграммы, созданные ранее.

Построим диаграмму последовательности для варианта использования «Обеспечить покупателя информацией» (рис. 4). Для этого добавим на диаграмму последовательности линии жизни и соотнесем объекты с актерами, иницилирующими вариант использования «Обеспечить покупателя информацией», и с необходимыми классами.

Для **добавления диаграммы последовательности в проект MS Visio** выполните следующие действия:

1. В проводнике по моделям найдите ветку «Основной пакет».
2. Нажмите по ней правой кнопкой мыши > Создать ...
3. В контекстном меню выберите пункт «Схема последовательностей».

Добавим сообщения, которыми обмениваются объекты для исполнения варианта использования.

Если объект имеет операцию (посмотреть в практическом занятии №8 «**Диаграммы классов**» наличие операции у класса, которому принадлежит объект).

1. Из группы фигур «**Последовательности UML**» добавить три фигуры типа «**Линия жизни объекта**». Для изменения названия необходимо дважды щелкнуть левой кнопкой мыши по фигуре. Откроется окно свойств объекта и, если в данном файле нет ранее созданных классов, окно создания нового класса. В данном примере необходимо создать три класса «**Товар**», «**Каталог товаров**» и «**Заказ**» и соответственно три объекта с такими же названиями.

2. С помощью поиска фигур найти фигуру «**Актер**» и добавить ее в рабочую область. Двойным щелчком левой кнопки мыши задать имя «**Клиент**».

3. Добавить фигуру «**Линия жизни**» и соедините ее начало с фигурой «**Клиент**».

4. Протянуть все линии жизни вниз листа.

5. Добавить фигуры «**Сообщение**» и соединить, руководствуясь следующими принципами:

5.1. Соединить фигурой «**Сообщение**» линию жизни клиента с линией жизни объекта товар. Двойным щелчком по сообщению открыть окно свойств и выбрать операцию **запросить товар**.

5.2. Соединить фигурой «**Сообщение**» линию жизни клиента с линией жизни объекта заказ. Двойным щелчком по сообщению открыть окно свойств и выбрать операцию **сформировать заказ**.

5.3. Соединить фигурой «**Сообщение**» линию жизни объекта товар с линией жизни объекта каталог товаров. Двойным щелчком по сообщению открыть окно свойств и выбрать операцию **проверить наличие**.

5.4. Соединить фигурой «**Сообщение (возврат)**» линию жизни объекта товар и линию жизни клиента. Двойным щелчком по сообщению открыть окно свойств и задать текст сообщения «**Предоставить информацию**».

6. Добавить фигуры «**Активация**» и расположить их на диаграмме в соответствии с рисунком 4.

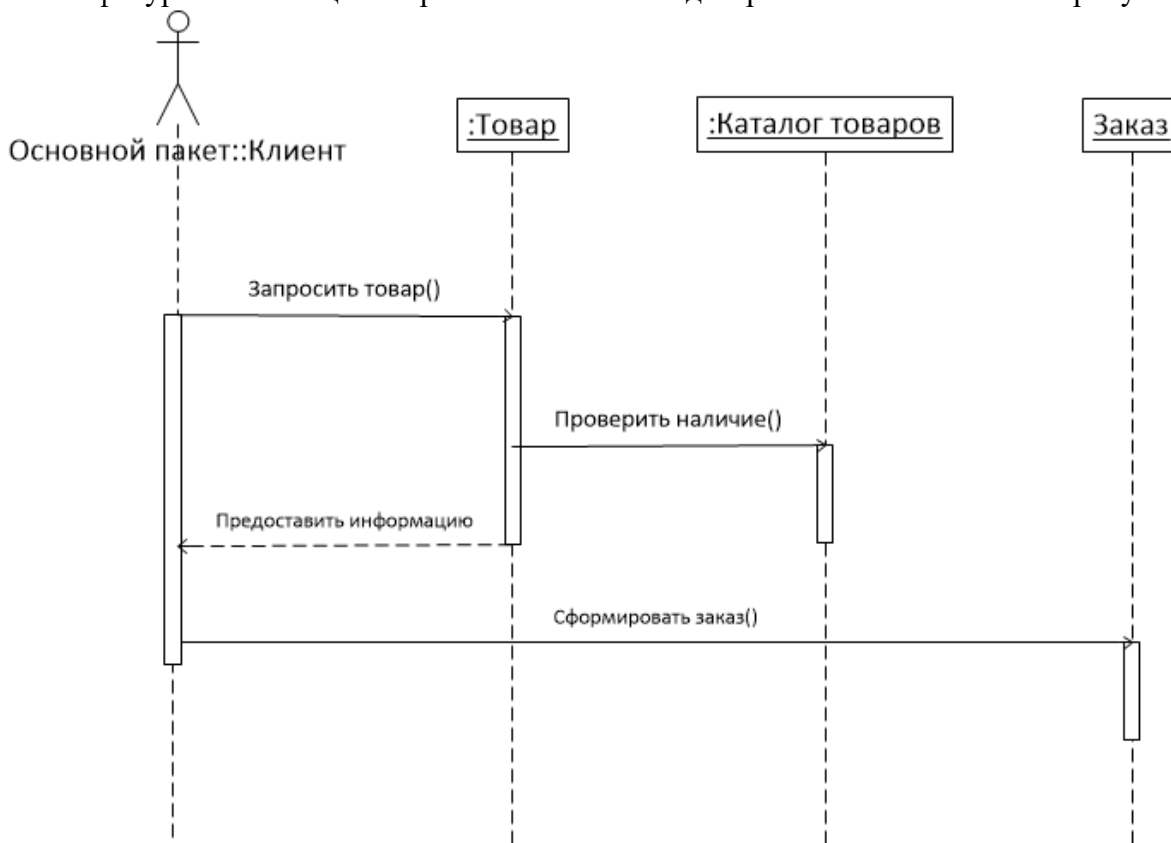


Рисунок 30 – Диаграмма последовательности для варианта использования «**Обеспечить покупателя информацией**»

При построении диаграмм последовательностей можно вносить коррективы в диаграмму классов. Если объект класса получает новую операцию, то она добавляется в соответствующий класс на диаграмме классов как метод.

Построим диаграмму последовательности для варианта использования «**Согласовать условия оплаты**» (рис. 5). Действия по построению диаграммы аналогичны построению диаграммы последовательности для варианта использования «**Обеспечить покупателя информацией**».

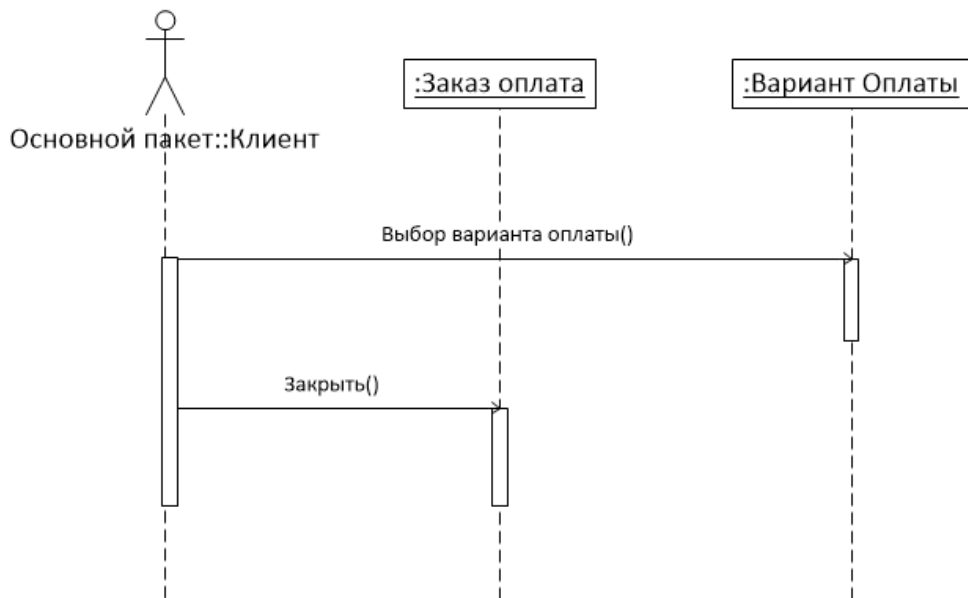


Рисунок 31 – Диаграмма последовательности для варианта использования «Согласовать условия оплаты»

Построим диаграмму последовательности для варианта использования «Заказать товар со склада» (рис. 6). Действия по построению диаграммы аналогичны построению предыдущих диаграмм последовательностей.

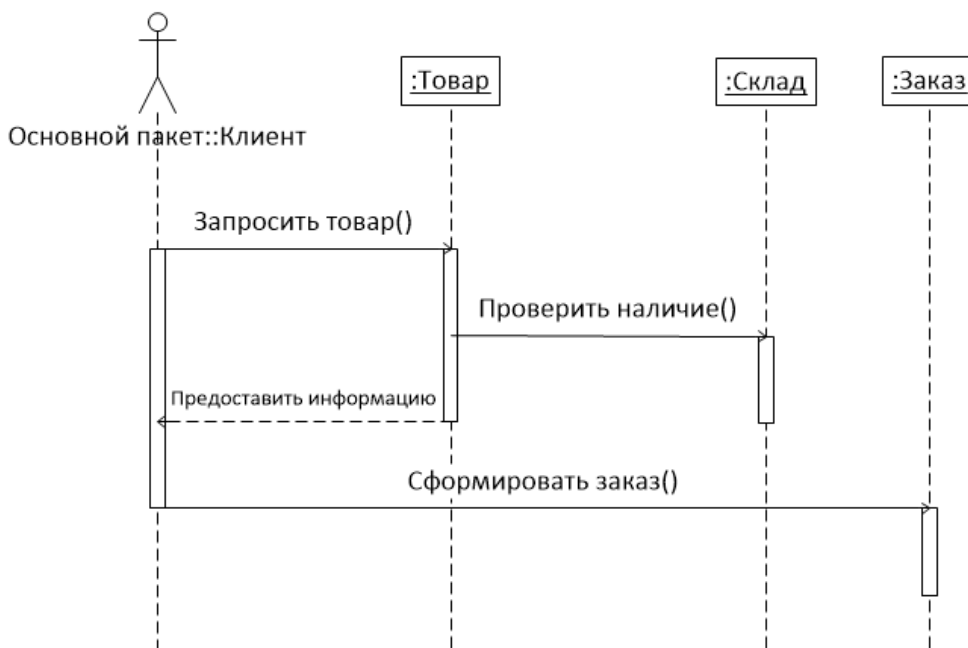


Рисунок 32 – Диаграмма последовательности для варианта использования «Заказать товар со склада»

Построим диаграмму последовательности для системы продажи товаров по каталогу (рис. 7).

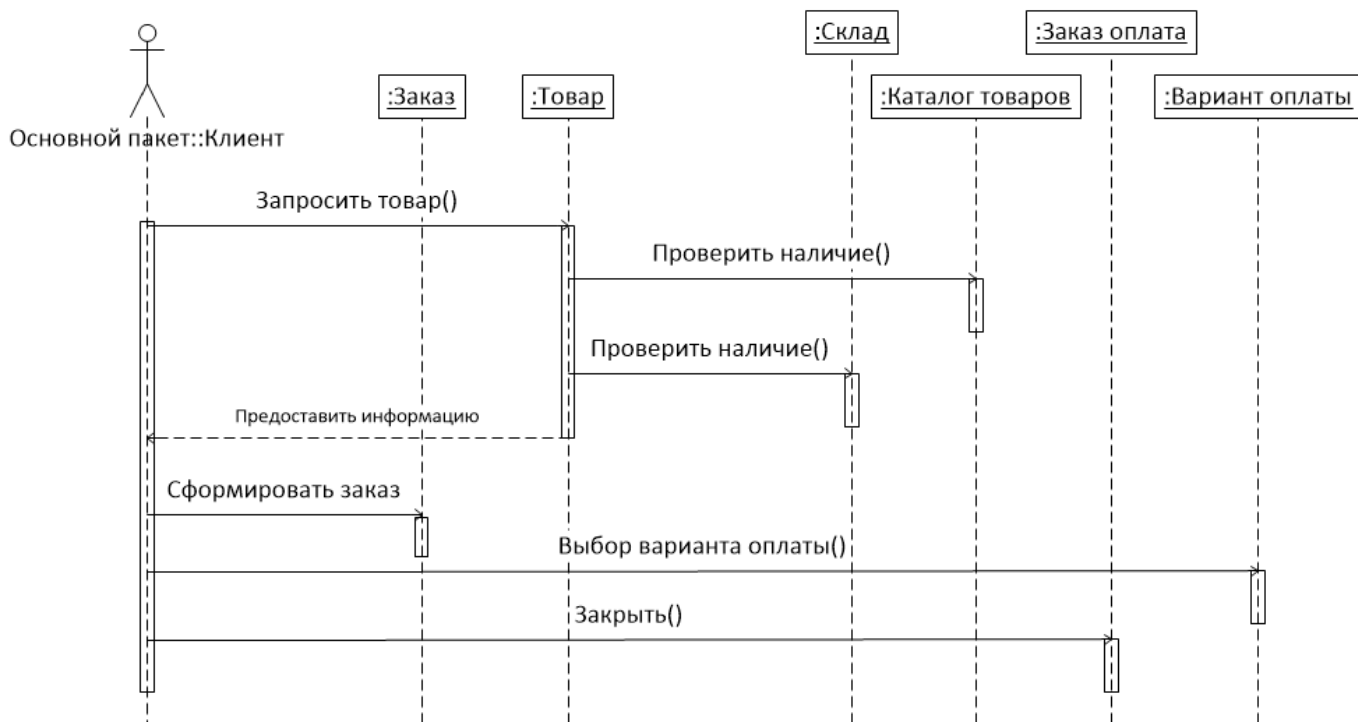


Рисунок 33 – Диаграмма последовательности для системы продажи товаров по каталогу

4 Задание

Построить диаграмму последовательности для каждого варианта использования, определенных в практическом занятии №8-9 и для всей системы в целом в соответствии с вариантом.

Отчет по практическому занятию выполняется в формате MS Word, который содержит пошаговое описание процесса построения диаграммы, а также скриншоты результатов согласно заданию.

5 Варианты

1. «Отдел кадров»;
2. «Агентство аренды»;
3. «Аптека»;
4. «Ателье»;
5. «Аэропорт»;
6. «Библиотека»;
7. «Кинотеатр»;
8. «Поликлиника»;
9. «Автосалон»;
10. «Таксопарк».
11. «Издательство»;
12. «Прокат велосипедов»;
13. «Спортивный клуб».

6 Контрольные вопросы

1. Для чего предназначена диаграмма последовательности?
2. Назовите и охарактеризуйте элементы диаграммы последовательности.
3. Что такое сообщение?
4. Что такое линия жизни?
5. Назовите виды сообщений.

Практическая работа №18

Разработка диаграмм прецедентов с помощью Visual Paradigm for UML

Цель: ознакомление с возможностями case-средства Visual Paradigm for UML Community Edition, освоение основных принципов создания диаграмм вариантов использования с помощью этого case-средства.

1 Задачи

Основными задачами практической работы являются:

- изучение среды case-средства Visual Paradigm for UML Community Edition;
- получить навыки создания диаграмм прецедентов.

2 Краткие теоретические сведения

1. Интерфейс Visual Paradigm for UML CE

Visual Paradigm for UML Community Edition – среда объектно-ориентированного проектирования на языке UML, распространяемая бесплатно для некоммерческого использования.

При первых запусках среда будет предлагать сообщить своё имя и e-mail для получения регистрационного кода. Получив код по электронной почте, следует активировать лицензию. Некоторые функции среды доступны только в её платных версиях. Начальное окно среды Visual Paradigm for UML Community Edition имеет вид, изображенный на рисунке 1.

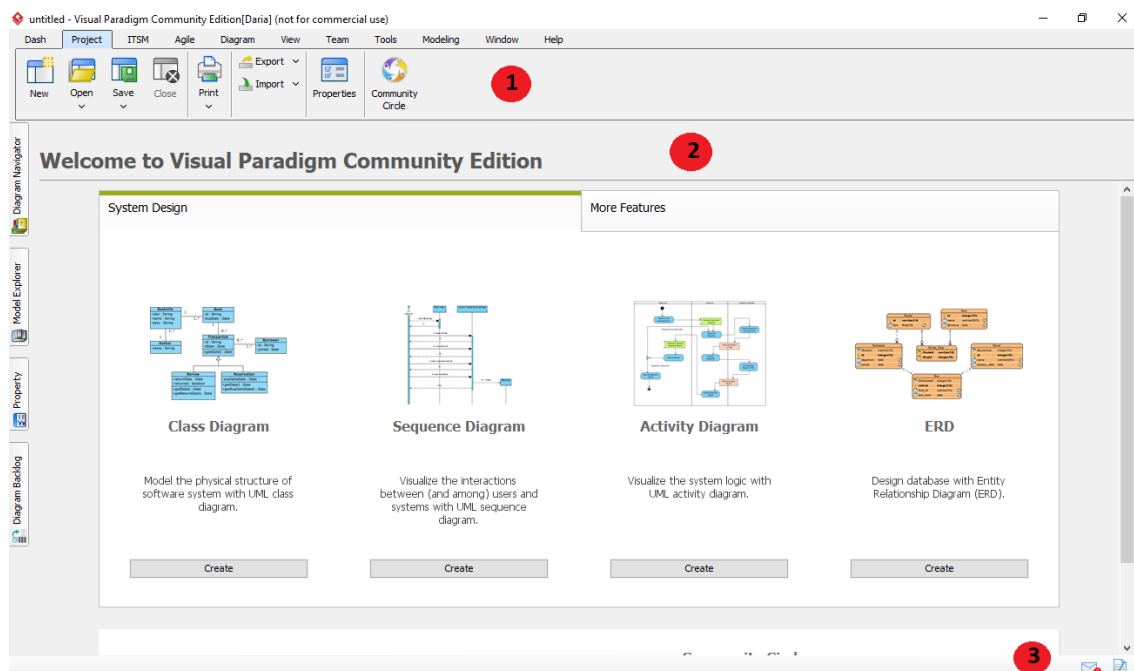


Рисунок 34 – Начальное окно Visual Paradigm for UML CE

В таблице 1 приведено описание основных элементов интерфейса среды объектно-ориентированного проектирования Visual Paradigm for UML Community Edition.

Таблица 1 – Описание элементов интерфейса Visual Paradigm for UML CE

№ п/п	Наименование	Описание
1	Панель инструментов	Панель со вкладками, позволяющая выполнять основные операции по созданию проектов и работе с диаграммами в Visual Paradigm
2	Редактор диаграмм	Рабочая область, внутри которой отображается проектируемая диаграмма
3	Строка состояния	Строка, в которой отображаются уведомления

Сохранение и открытие проектов

Для сохранения созданного проекта необходимо выбрать в панели инструментов *Project > Save* или *Project > Save as...* При первом сохранении проекта, среда проектирования предложит выбрать место для сохранения.

Для открытия ранее созданного проекта необходимо выбрать в панели инструментов *Project > Open* и указать путь к файлу проекта, имеющему расширение *Visual Paradigm Project (*.vpp)*.

Основы работы с диаграммами в Visual Paradigm for UML CE

Далее описываются основные шаги по созданию диаграмм, добавлению фигур и установлению связей между ними.

1. Создание диаграмм

Рассмотрим в качестве примера процесс создания диаграммы вариантов использования. Для создания диаграммы прецедентов необходимо выполнить следующую последовательность действий.

- 1) Выбрать *Diagram > New* в панели инструментов (рис. 2).
- 2) Начать набор названия диаграммы в строке поиска, в данном случае *use case diagram*.

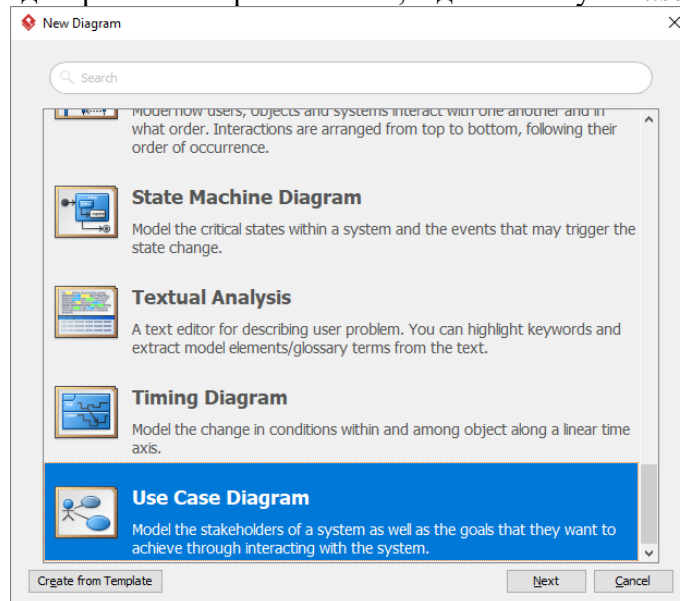


Рисунок 35 – Окно создания новой диаграммы

- 3) Выбрать диаграмму в списке и нажать кнопку *Next*.
- 4) Задать имя диаграммы, место расположения и описание.
- 5) Нажать кнопку *Ok*.

2. Создание и соединение фигур

2.1. Используя панель инструментов для работы с диаграммами

Далее рассмотрим процесс добавления фигуры *Актер* с помощью панели инструментов для работы с диаграммами (рис. 3).

- 1) Нажать на кнопку с изображением *Актера* в панели инструментов для работы с диаграммами.
- 2) Щелкнуть в рабочей области диаграммы чтобы создать актера и ввести его название.
- 3) Нажать в произвольном месте рабочей области либо нажать клавишу *Enter* на клавиатуре.



Рисунок 36 – Добавление фигуры Актер

2.2. Используя каталог ресурсов

Переместив курсор мыши на фигуру в ее правом верхнем углу появляется значок, который предназначен для доступа к каталогу ресурсов.

Каталог ресурсов позволяет создавать новую фигуру, которая автоматически соединяется с существующей. Также его можно использовать для создания связи между двумя существующими фигурами.

Рассмотрим последовательность действий для создания варианта использования от существующего актера (рис. 4).

- 1) Переместить курсор мыши на фигуру *Актер*.
- 2) Нажать на кнопку *Каталог ресурсов* в правом верхнем углу и потянуть указатель вправо.
- 3) Отпустить кнопку мыши и выбрать *Association > Use Case*.
- 4) Ввести название варианта использования.

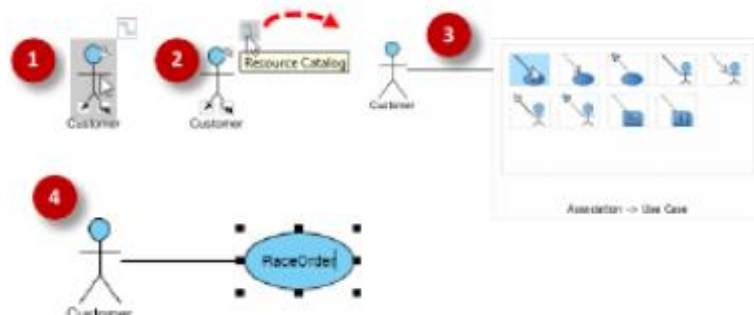


Рисунок 37 – Добавление варианта использования с помощью каталога ресурсов

3. Изменение размера фигур

После нажатия на фигуру появляется несколько обработчиков изменения размера. Перетаскивая их можно изменять размер фигуры.

4. Добавление контрольных точек к соединениям

Для большинства диаграмм в качестве соединения по умолчанию используется «наклонное соединение», представляющее собой прямую линию. Добавление контрольных точек позволит изменить положение линии (рис. 5).

Для добавления контрольной точки необходимо навести курсор мыши на соединение нажать левую кнопку мыши и не отпуская ее переместить точку в нужное место.

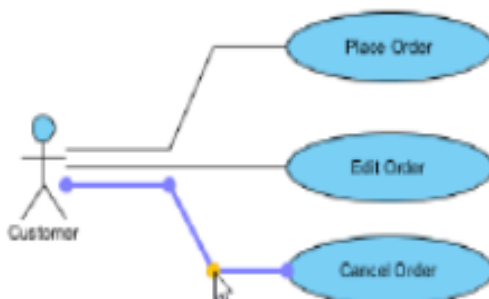


Рисунок 38 – Добавление контрольной точки

Примечание. Существует 5 типов соединений. Чтобы применить другой тип необходимо щелкнуть правой кнопкой мыши по линии и выбрать *Styles and Formatting > Connector Styles* и выбрать подходящий тип. Если необходимо изменить сразу все соединения нужно щелкнуть правой кнопкой мыши в свободном месте рабочей области и выбрать *Connectors* во всплывающем меню.

5. Изменение цвета фигуры

Для придания более выразительного внешнего вида диаграмме можно изменять цвет фигур.

Рассмотрим последовательность действий для изменения цвета варианта использования.

1) Щелкнуть правой кнопкой мыши по фигуре *Вариант использования* и в контекстном меню выбрать *Styles and Formatting > Formats...*

2) Открыть вкладку *Backgrounds* в окне *Formats*. Выбрать нужный цвет и нажать кнопку *Ok* чтобы применить выбранный цвет к фигуре.

Экспорт диаграммы

Готовую диаграмму можно полностью или частично копировать в формате изображения JPG или EMF в буфер обмена для экспорта в другое приложение.

Для этого следует выделить нужные объекты или всю диаграмму (Ctrl+A) и в контекстном меню выбрать команду *Copy > Copy to Clipboard as Image (JPG)*. Данная команда дублируется комбинацией клавиш Ctrl+Alt+C. После выполненных действий изображение диаграммы готово к вставке в другом приложении.

3 Методика выполнения

В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

1. Запустите visual Paradigm for UML CE.
2. Создайте новый проект: *Project > New*. Введите название проекта и нажмите кнопку *Create Blank Project* (рис. 6).

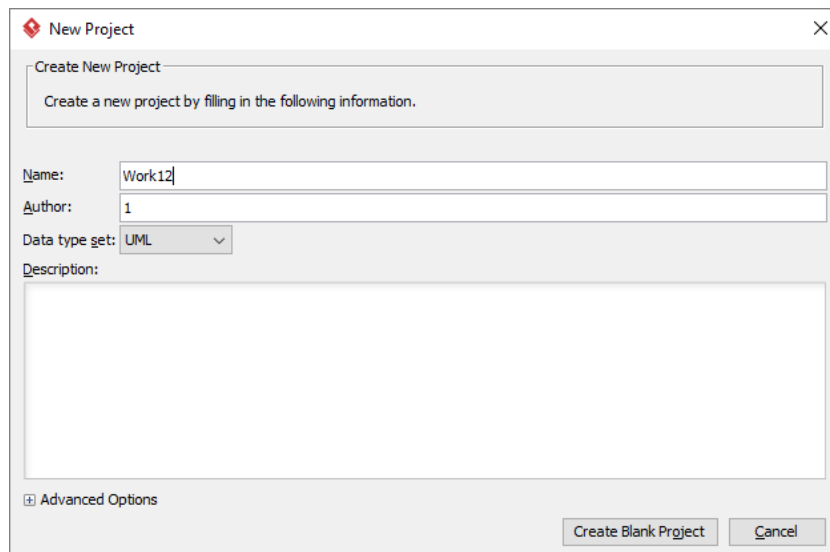


Рисунок 39 – Окно создания проекта

3. Создайте новую диаграмму вариантов использования (*Use Case Diagram*): *Diagram > New* (рис. 7).

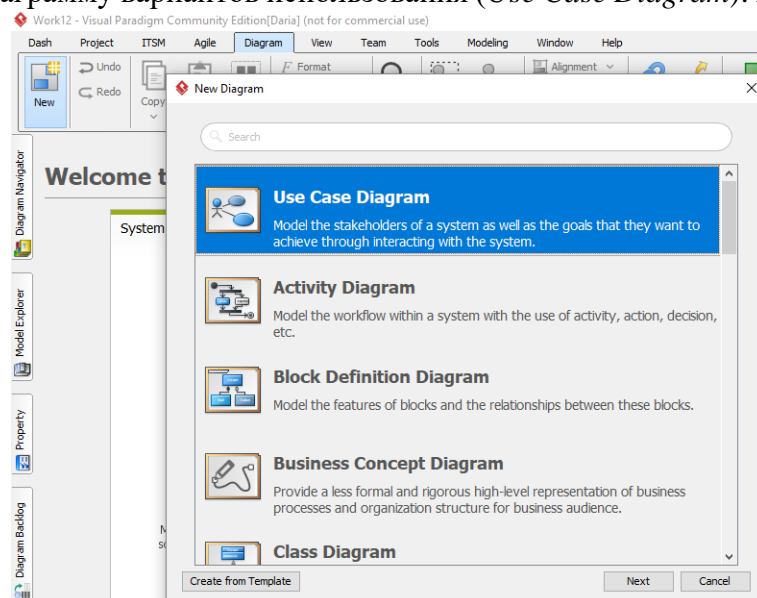


Рисунок 40 – Добавление диаграммы в проект

4. Присвойте имя диаграмме: *SaleOfGoods* (рис. 8).

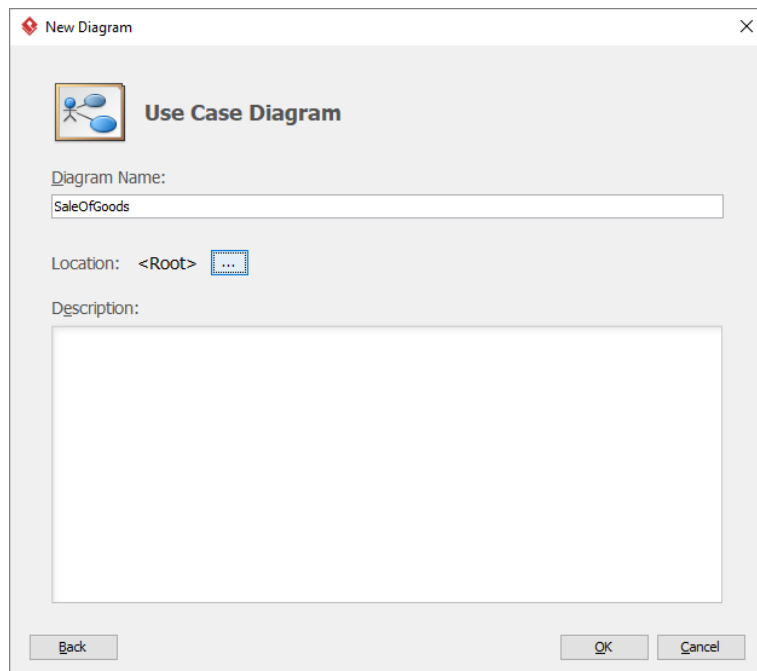


Рисунок 41 – Указание имени диаграммы

5. Добавьте фигуру *System*. Для этого в панели инструментов для работы с фигурами выберите нужную фигуру и щелкните в любом месте рабочей области. После этого переименуйте систему задав название: «Система продажи товаров по каталогу» (рис. 9).

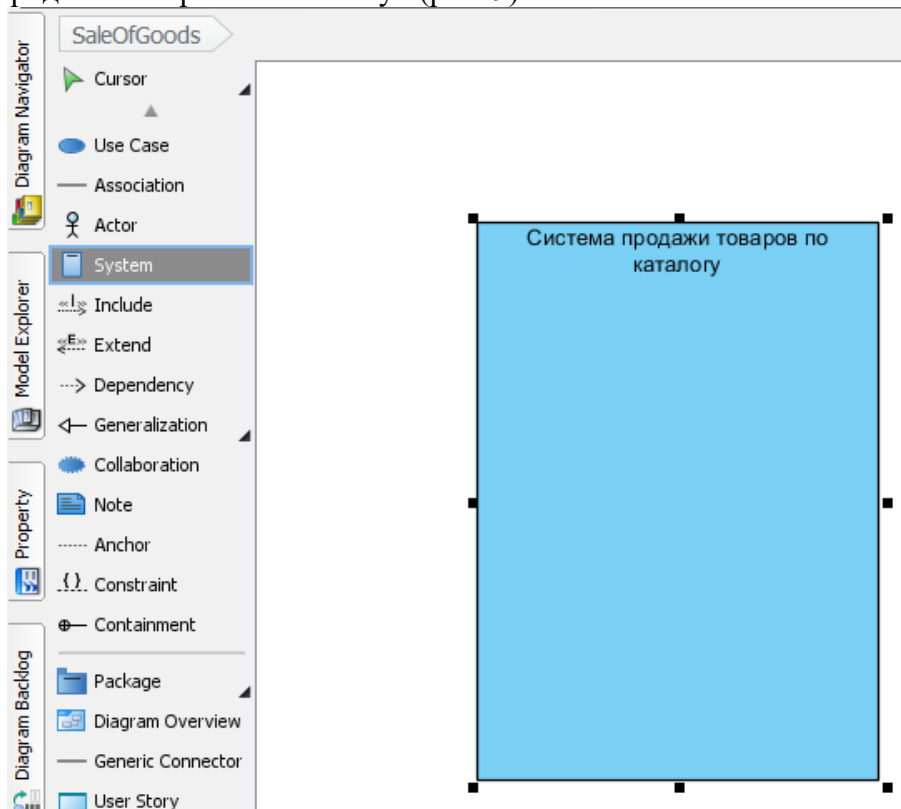


Рисунок 42 – Добавление элемента Система

6. Далее добавьте актеров *Продавец* и *Покупатель* и вариант использования *Оформить заказ на покупку товара*.

7. Добавьте связь актеров с вариантом использования. Для этого: выделите актера > в правом верхнем углу откройте каталог ресурсов > в контекстном меню выберите use case > во вспомогательном окне начните вводить название ранее созданного прецедента «Оформить заказ на покупку товара» > нажмите кнопку *Ok*.

Результат выполнения действий 6 и 7 показан на рисунке 10.



Рисунок 43 – Исходная диаграмма прецедентов

8. Далее необходимо установить кратность для отношения ассоциации между актерами и вариантом использования. Для этого необходимо навести курсор на связь и дважды щелкнуть левой кнопкой мыши по точке вначале линии. В появившемся контекстном окне необходимо задать кратность данной стороны отношения (рис. 11).

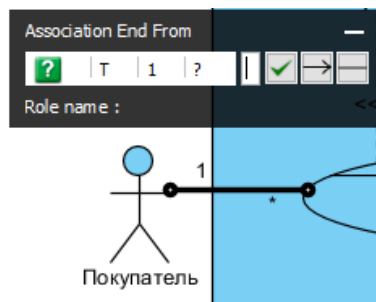


Рисунок 44 – Задание кратности отношения

Для задания кратности второй стороне отношения нажмите кнопку со стрелкой вправо и проделайте те же действия. Для того, чтобы применить произведенные действия достаточно нажать кнопку с зеленой галочкой либо щелкнуть в любом пустом месте рабочей области.

9. Дополним исходную диаграмму. Для этого добавим 4 варианта использования «Обеспечить покупателя информацией», «Согласовать условия оплаты», «Заказать товар со склада» и «Запросить каталог товаров» (рис. 12).

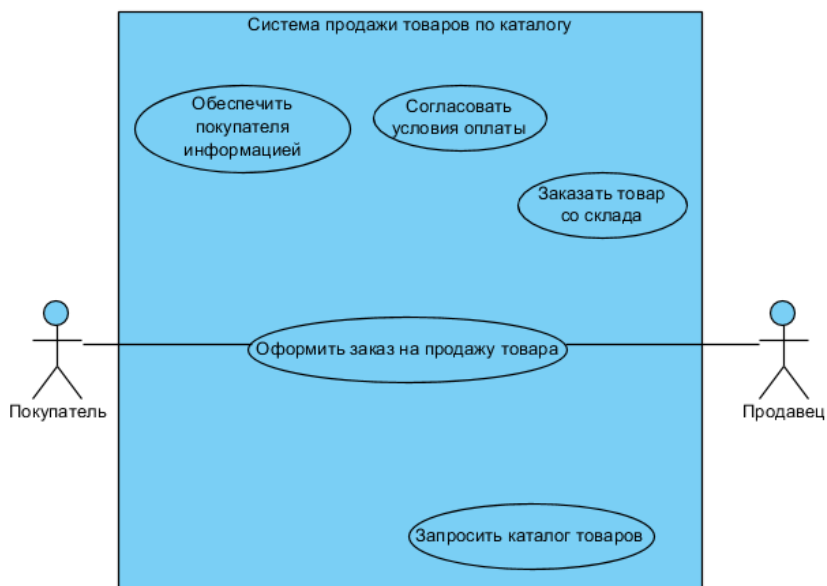


Рисунок 45 – Дополнительные варианты использования

10. Установим связи между основным и дополнительными вариантами использования. «Запросить каталог товаров» связан с «Оформить заказ на продажу товара» отношением расширения. Для того чтобы установить этот тип соединения необходимо в панели инструментов для работы с диаграммами выбрать соответствующую фигуру (рис. 13).

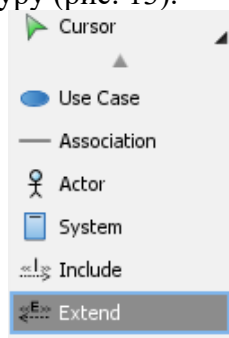


Рисунок 46 – Фигура отношения расширения

После чего соединить два варианта использования.

«Обеспечить покупателя информацией», «Согласовать условия оплаты», «Заказать товар со склада» связаны с «Оформить заказ на покупку товара» отношением включения. Для того чтобы установить этот тип соединения необходимо в панели инструментов для работы с диаграммами выбрать соответствующую фигуру (рис. 13).

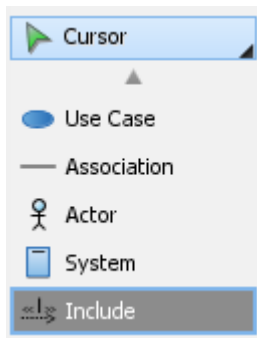


Рисунок 47 – Фигура отношения включения

После чего соединить варианты использования «Обеспечить покупателя информацией», «Согласовать условия оплаты», «Заказать товар со склада» с «Оформить заказ на покупку товара».

В результате выполнения всех вышеописанных действий диаграмма вариантов использования должна иметь вид, показанный на рисунке 14.

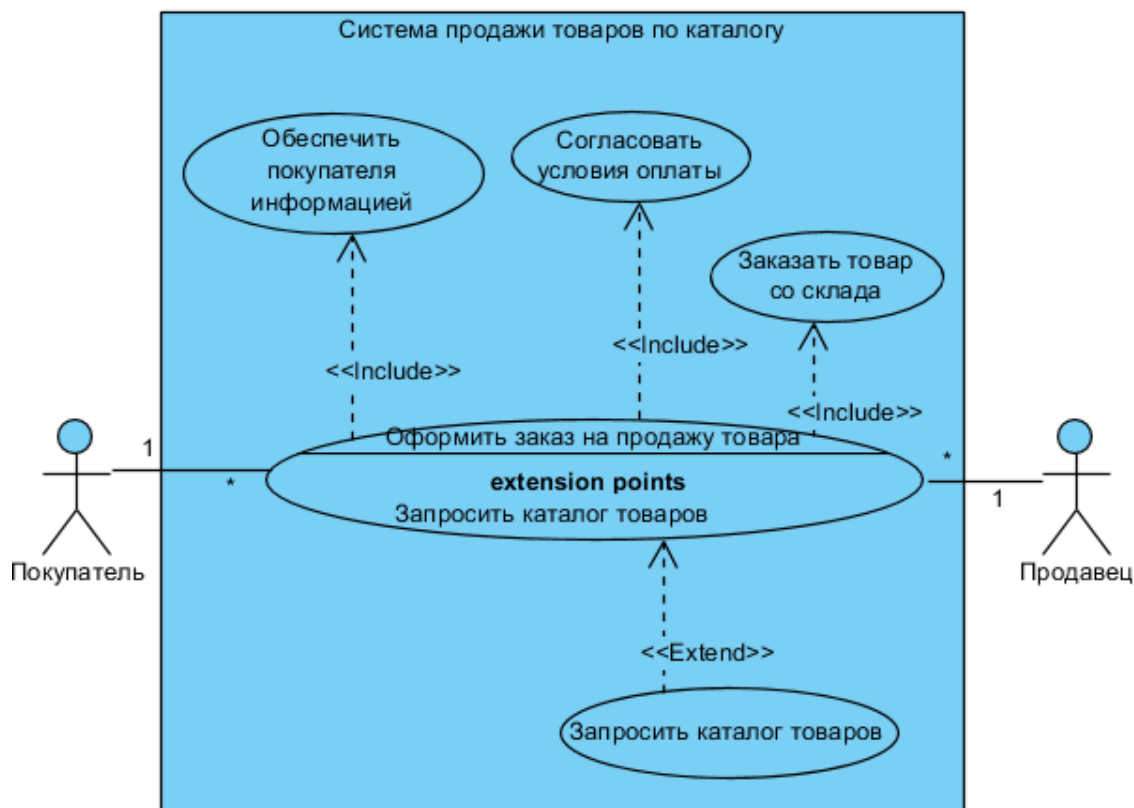


Рисунок 48 – Дополненная диаграмма вариантов использования

4 Задание

Построить диаграмму прецедентов (вариантов использования) в соответствии с вариантом.

Отчет по практическому занятию выполняется в формате MS Word, который содержит пошаговое описание процесса построения организационной диаграммы, а также скриншоты результатов согласно заданию.

5 Варианты

1. «Отдел кадров»;
2. «Агентство аренды»;
3. «Аптека»;
4. «Ателье»;
5. «Аэропорт»;
6. «Библиотека»;
7. «Кинотеатр»;
8. «Поликлиника»;
9. «Автосалон»;
10. «Таксопарк».
11. «Издательство»;
12. «Прокат велосипедов»;
13. «Спортивный клуб».

6 Контрольные вопросы

1. Для чего используется диаграмма вариантов использования (прецедентов) Use Case?
2. Назовите основные элементы диаграммы прецедентов.
3. Как создать диаграмму вариантов использования с помощью Visual Paradigm for UML CE?хм

Практическая работа №19

Разработка диаграмм классов с помощью Visual Paradigm for UML

Цель: изучение основ создания диаграмм классов на языке UML с помощью case-средства Visual Paradigm for UML CE, применение приобретенных навыков для построения объектно-ориентированных моделей определенной предметной области.

1 Задачи

Основными задачами практической работы являются:

- получить навыки создания диаграмм классов с помощью case-средства Visual Paradigm for UML CE.

2 Краткие теоретические сведения

Диаграмма классов отражает различные взаимосвязи между отдельными сущностями предметной области, а также описывает их внутреннюю структуру и типы отношений.

Диаграмма классов (class diagram) – диаграмма языка UML, на которой представлена совокупность декларативных или статических элементов модели, таких как классы с атрибутами и операциями, а также связывающие их отношения.

Диаграмма классов предназначена для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования.

Диаграмма классов содержит интерфейсы, пакеты, отношения и даже отдельные экземпляры классификаторов, такие как объекты и связи.

Класс (class) – абстрактное описание множества однородных объектов, имеющих одинаковые атрибуты, операции и отношения с объектами других классов.

Графически изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции (рис. 1). В этих секциях могут указываться имя класса, атрибуты и операции класса. Даже если секции атрибутов и операций пусты, в обозначении класса они должны быть выделены горизонтальной линией.

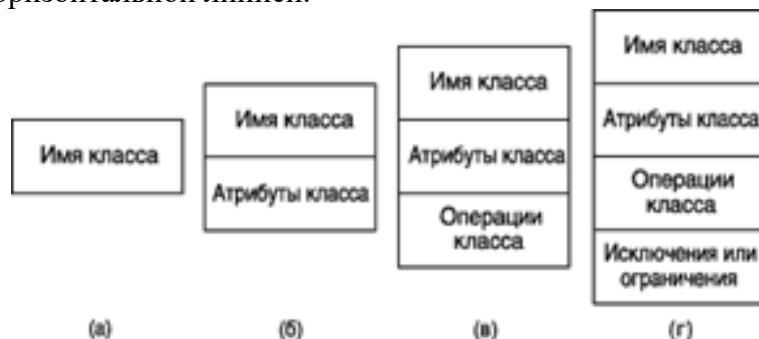


Рисунок 49 – Варианты графического изображения класса на диаграмме классов

В первом случае для класса Окружность (рис. 2, а) указаны только его атрибуты – точка на координатной плоскости, которая определяет расположение ее центра. Для класса Окно (рис. 2, б) указаны только его операции, при этом секция его атрибутов оставлена пустой. Для класса Счет (рис. 2, в) дополнительно изображена четвертая секция, в которой указано требование – реализовать резервное копирование объектов этого класса.

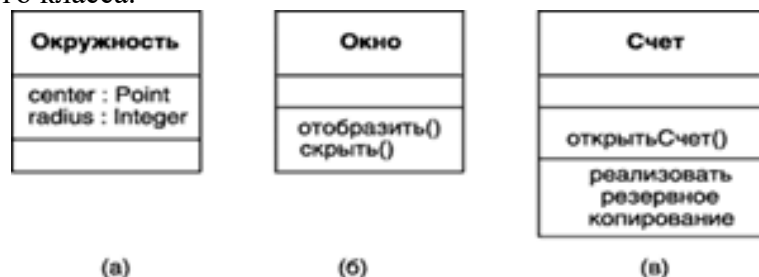


Рисунок 50 – Примеры графического изображения конкретных классов

Имя класса должно быть уникальным в пределах пакета, который может содержать одну или несколько диаграмм классов. Имя класса записывается по центру секции имени полужирным шрифтом и должно начинаться с заглавной буквы.

В секции имени класса могут также находиться стереотипы или ссылки на стандартные шаблоны, от которых образован данный класс и, соответственно, от которых он наследует атрибуты и операции. Здесь также могут записываться и другие общие свойства этого класса.

Класс может иметь или не иметь экземпляров или объектов. В зависимости от этого в **языке UML различают** конкретные и абстрактные классы.

Конкретный класс (concrete class) – класс, на основе которого могут быть непосредственно созданы экземпляры или объекты.

Рассмотренные выше обозначения относятся к конкретным классам.

Абстрактный класс (abstract class) – класс, который не имеет экземпляров или объектов.

Для обозначения имени абстрактного класса используется курсив.

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

Атрибут класса служит для представления отдельного свойства или признака, который является общим для всех объектов данного класса. Каждому атрибуту класса соответствует отдельная строка текста, которая состоит из квантора видимости атрибута, имени атрибута, его кратности, типа значений атрибута и, возможно, его исходного значения.

Общий формат записи отдельного атрибута класса следующий:

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

Видимость (visibility) – качественная характеристика описания элементов класса, характеризующая потенциальную возможность других объектов модели оказывать влияние на отдельные аспекты поведения данного класса.

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

– "+" – область видимости типа общедоступный (public). Атрибут с этой областью видимости доступен или виден из любого другого класса пакета, в котором определена диаграмма.

– "#" – область видимости типа защищенный (protected). Атрибут недоступен или невиден для всех классов, за исключением подклассов данного класса.

– "-" – область видимости типа закрытый (private). Атрибут недоступен или невиден для всех классов без исключения.

– "~" – область видимости типа пакетный (package). Атрибут недоступен или невиден для всех классов за пределами пакета, в котором определен класс-владелец данного атрибута.

Вместо условных графических обозначений можно записывать соответствующее ключевое слово: public, protected, private, package.

Имя атрибута используется в качестве идентификатора соответствующего атрибута и поэтому должно быть уникальным в пределах данного класса. Имя атрибута - обязательный элемент, должно начинаться со строчной (малой) буквы и не должно содержать пробелов.

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

Тип атрибута представляет собой выражение, семантика которого определяется некоторым типом данных, определенным в пакете Типы данных языка UML или самим разработчиком, или в зависимости от языка программирования, который предполагается использовать для реализации данной модели.

Исходное значение служит для задания начального значения соответствующего атрибута в момент создания отдельного экземпляра класса. Если оно не указано, то значение соответствующего атрибута не определено на момент создания нового экземпляра класса.

Пояснительный текст в фигурных скобках может означать две различные конструкции. Если в этой строке имеется знак равенства, то вся запись 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.

Знак "/" перед именем атрибута указывает на то, что данный атрибут является производным от некоторого другого атрибута этого же класса.

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.

Общий формат записи отдельной операции класса следующий:

<квантор видимости> <имя операции>(список параметров):

<выражение типа возвращаемого значения> {строка-свойство}

Квантор видимости операции аналогичен квантору видимости атрибутов.

Имя операции – обязательный элемент, должно начинаться со строчной (малой) буквы, и записываться без пробелов.

Список параметров является перечнем разделенных запятой формальных параметров, каждый из которых, в свою очередь, может быть представлен в следующем виде:

<направление параметра> <имя параметра> : <выражение типа> = <значение параметра по умолчанию>

Параметр (parameter) – спецификация переменной операции, которая может быть изменена, передана или возвращена.

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

Кроме внутреннего устройства классов важную роль при разработке проектируемой системы имеют различные отношения между классами, которые также могут быть изображены на диаграмме классов. Базовые отношения, изображаемые на диаграммах классов:

- отношение ассоциации (association relationship);
- отношение обобщения (generalization relationship);
- отношение агрегации (aggregation relationship);
- отношение композиции (composition relationship).

Отношение ассоциации соответствует наличию произвольного отношения или взаимосвязи между классами. Оно обозначается сплошной линией со стрелкой или без нее и с дополнительными символами, которые характеризуют специальные свойства ассоциации.

В качестве простого примера *ненаправленной бинарной ассоциации* можно рассмотреть отношение между двумя классами – классом «Компания» и классом «Сотрудник» (рис. 3). Они связаны между собой бинарной ассоциацией «Работает». Для данного отношения определен следующий порядок чтения следования классов - сотрудник работает в компании.



Рисунок 51 – Графическое изображение ненаправленной бинарной ассоциации между классами

Направленная бинарная ассоциация изображается сплошной линией с простой стрелкой на одной из ее концевых точек. Направление этой стрелки указывает на то, какой класс является первым, а какой – вторым.

В качестве простого примера направленной бинарной ассоциации можно рассмотреть отношение между двумя классами – классом «Клиент» и классом «Счет» (рис. 4). Они связаны между собой бинарной ассоциацией с именем «Имеет», для которой определен порядок следования классов. Это означает, что конкретный объект класса «Клиент» всегда должен указываться первым при рассмотрении взаимосвязи с объектом класса «Счет», например, <клиент, счет_1, счет_2, ..., счет_n>.

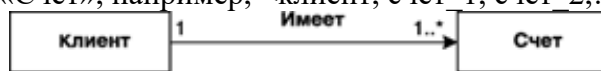


Рисунок 52 – Графическое изображение направленной бинарной ассоциации между классами

Частный случай отношения ассоциации – так называемая *исключающая ассоциация* (Xor-association). Семантика данной ассоциации указывает на то, что из нескольких потенциально возможных вариантов данной ассоциации в каждый момент времени может использоваться только один.

На диаграмме классов исключающая ассоциация изображается пунктирной линией, соединяющей две и более ассоциации (рис. 5), рядом с которой записывается ограничение в форме строки текста в фигурных скобках: {xor}.

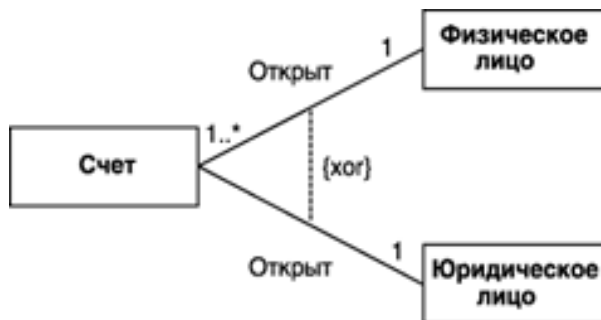


Рисунок 53 – Графическое изображение исключающей ассоциации между тремя классами

Отношение обобщения является отношением классификации между более общим элементом (родителем или предком) и более частным или специальным элементом (дочерним или потомком).

Менее общий элемент модели должен быть согласован с более общим элементом и может содержать дополнительную информацию. Отношение обобщения описывает иерархическое строение классов и наследование их свойств и поведения.

Согласно одному из главных принципов методологии ООП – наследованию, класс-потомок обладает всеми свойствами и поведением класса-предка, а также имеет собственные свойства и поведение, которые могут отсутствовать у класса-предка.

Отношение обобщения обозначается сплошной линией с треугольной стрелкой на одном из концов (рис. 6). Стрелка указывает на более общий класс (класс-предок или суперкласс), а ее начало – на более специальный класс (класс-потомок или подкласс).



Рисунок 54 – Графическое изображение отношения обобщения в языке UML

От одного класса-предка одновременно могут наследовать несколько классов-потомков. т.е. в класс-предок входит несколько линий со стрелками.

Пример: класс «Транспортное средство» (курсив обозначает абстрактный класс) может выступать в качестве суперкласса для подклассов, соответствующих конкретным транспортным средствам, таким как: «Автомобиль», «Автобус», «Трактор» и другим (рис 7).



Рисунок 55 – а) Пример графического изображения отношения обобщения для нескольких классов-потомков; б) Альтернативный вариант графического изображения отношения обобщения классов для случая объединения отдельных линий

Отношение агрегации имеет место между несколькими классами в том случае, если один из классов представляет собой сущность, которая включает в себя в качестве составных частей другие сущности. Данное отношение применяется для представления системных взаимосвязей типа «часть-целое» и показывает, из каких элементов состоит система, и как они связаны между собой.

Очевидно, что рассматриваемое в таком аспекте деление системы на составные части представляет собой иерархию, но принципиально отличную от той, которая порождается отношением обобщения. Отличие заключается в том, что части системы никак не обязаны наследовать ее свойства и поведение, поскольку являются самостоятельными сущностями. Более того, части целого обладают собственными атрибутами и операциями, которые существенно отличаются от атрибутов и операций целого. Графически отношение агрегации изображается сплошной линией, один из концов которой представляет собой не закрашенный внутри ромб. Этот ромб указывает на тот класс, который представляет собой «целое» или класс-контейнер. Остальные классы являются его «частями» (рис. 8).

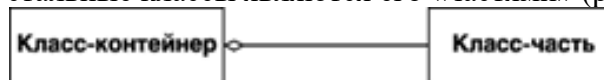


Рисунок 56 – Графическое изображение отношения агрегации в языке UML

В качестве примера отношения агрегации можно рассмотреть взаимосвязь типа «часть-целое», которая имеет место между классом «Системный блок» персонального компьютера и его составными частями: «Процессор», «Материнская плата», «Оперативная память», «Жесткий диск» и «Дисковод гибких дисков» (рис. 9).



Рисунок 57 – Диаграмма классов для иллюстрации отношения агрегации на примере системного блока ПК

Отношение композиции – частный случай отношения агрегации, и служит для спецификации более сильной формы отношения «часть-целое», при которой составляющие части тесно взаимосвязаны с целым. Особенность: с уничтожением целого уничтожаются и все его составные части.

Пример (рис. 10): Программное средство представляет собой базу данных «Автоматизация процесса составления расписания в учебном заведении». Программное средство обеспечивает корректировку данных, а именно в БД «Группы» может изменяться перечень предметов в соответствии с курсом группы и отделением; в БД «Предметы» могут изменяться номера аудиторий и фамилии преподавателей; осуществляет поиск по ФИО преподавателя, номеру аудитории, названию предмета и номеру группы. Программное средство составляет расписание работы для конкретного преподавателя на неделю; для

группы на неделю; для группы по конкретному предмету, а также отчет о загрузке аудиторий на каждый день.

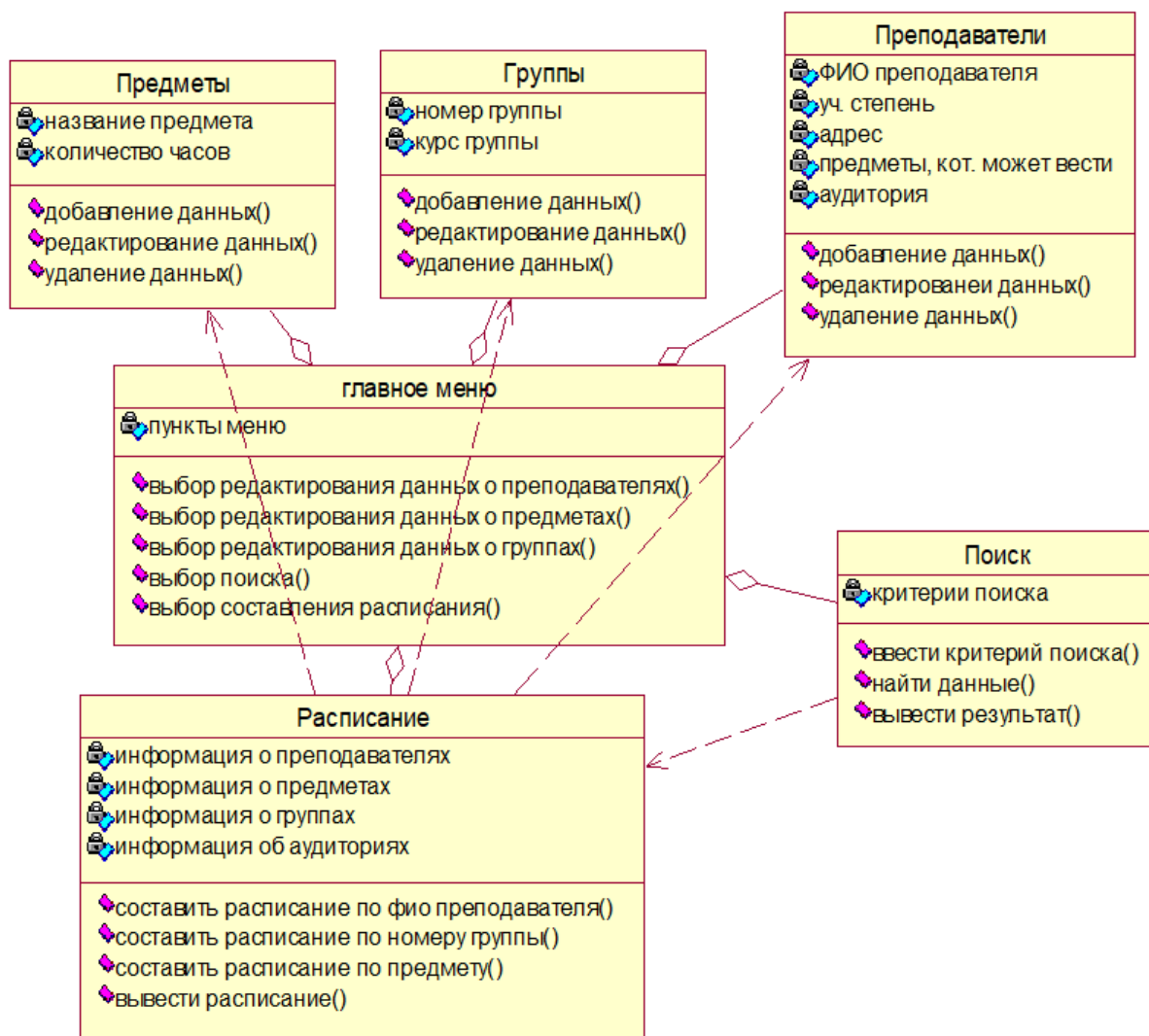


Рисунок 58 – Пример диаграммы классов

3 Методика выполнения

В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

1. Запустите Visual Paradigm for UML CE.
2. Откройте проект, содержащий диаграмму вариантов использования, построенную в практическом занятии №12.
3. Для добавления новой диаграммы достаточно в меню Diagram > New Diagram выбрать тип диаграммы Class Diagram и нажмите кнопку Next.
4. Задайте имя диаграммы SaleSystemClassDiagram. Нажмите ОК.
5. В результате отобразится пустая рабочая область с элементами для построения диаграммы классов (рис. 11).

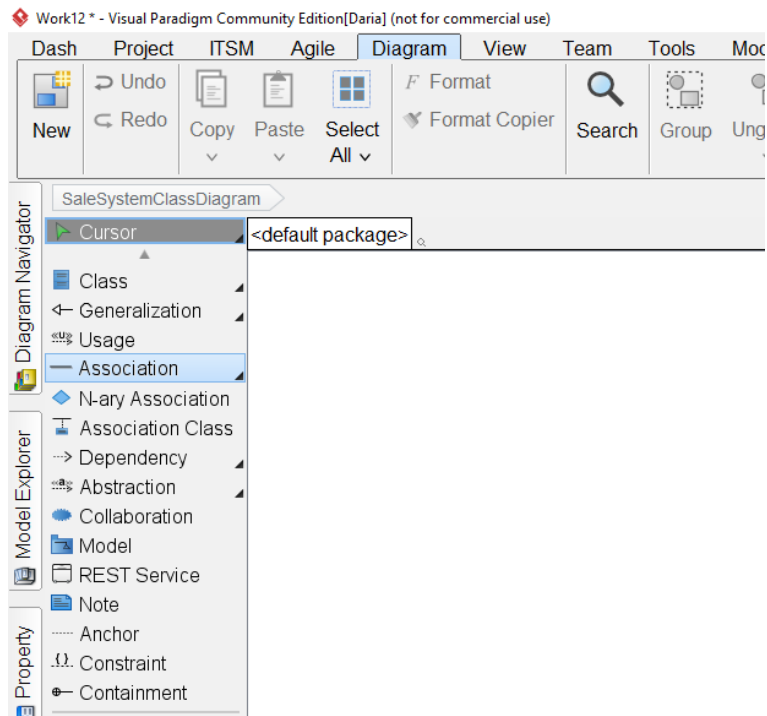


Рисунок 59 – Рабочая область для построения диаграммы классов

Далее будут описаны принципы построения диаграммы классов.

Для создания класса необходима на панели инструментов выбрать Class и затем курсором мыши выбрать место расположения на диаграмме (Рисунок 12). После создания класса необходимо задать его имя (Рисунок 13).

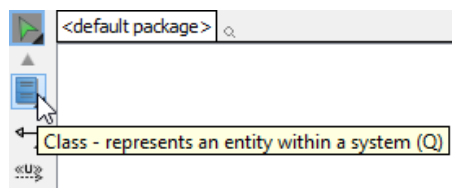


Рисунок 60 – Создание класса

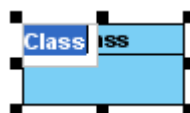


Рисунок 61 – Задание имени в созданном классе

Для создания ассоциативной связи между классами выберите **Association > Class** (Рисунок 14).

Перетащите курсор мыши на пустое место на диаграмме для создания нового или на уже существующий класс для соединения. Отпустите кнопку мыши (Рисунок 15).

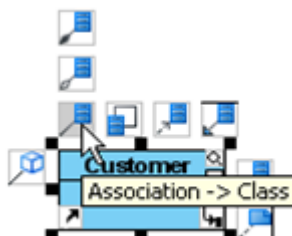


Рисунок 62 – Выбор типа связи



Рисунок 63 – Создание ассоциативной связи

Для создания связи агрегации между классами выберите **Aggregation -> Class** (Рисунок 19).

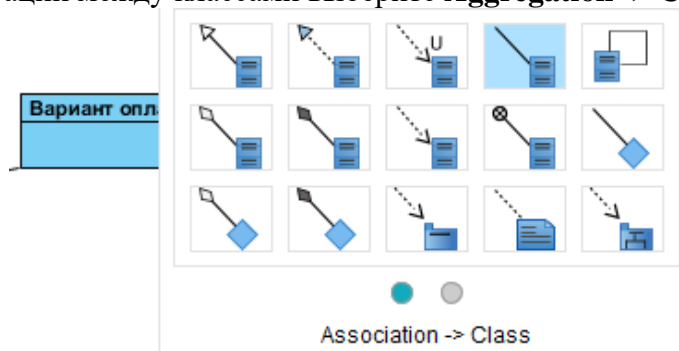


Рисунок 64 – Создание связи агрегации

Чтобы изменить кратность ассоциативной связи, щелкните правой кнопкой мыши на конце ассоциации, выберите **Multiplicity** из всплывающего меню, а затем выберите кратность (Рисунок 17). Либо выделите связи и нажмите **Enter** (рисунок 18).

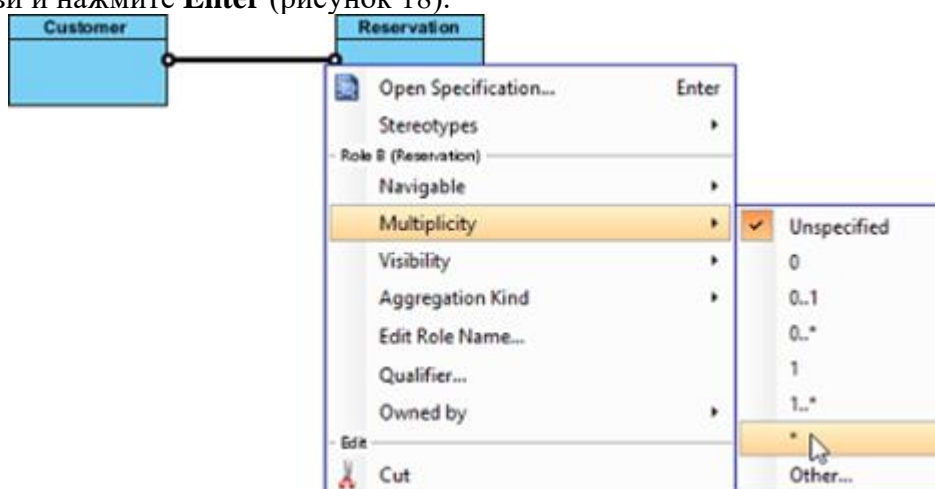


Рисунок 65 – Изменение кратности связи

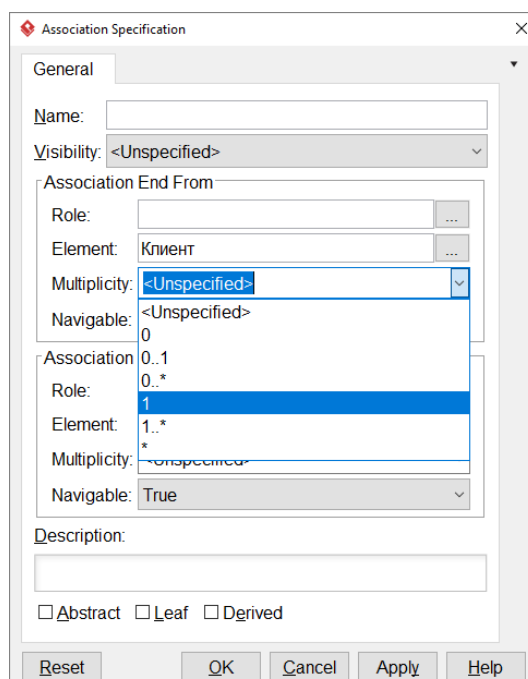


Рисунок 66 – Изменение кратности связи

Чтобы показать направление ассоциации, щелкните правой кнопкой мыши на связи и выберите **Presentation Options > Show Direction** из всплывающего меню (Рисунок 19).

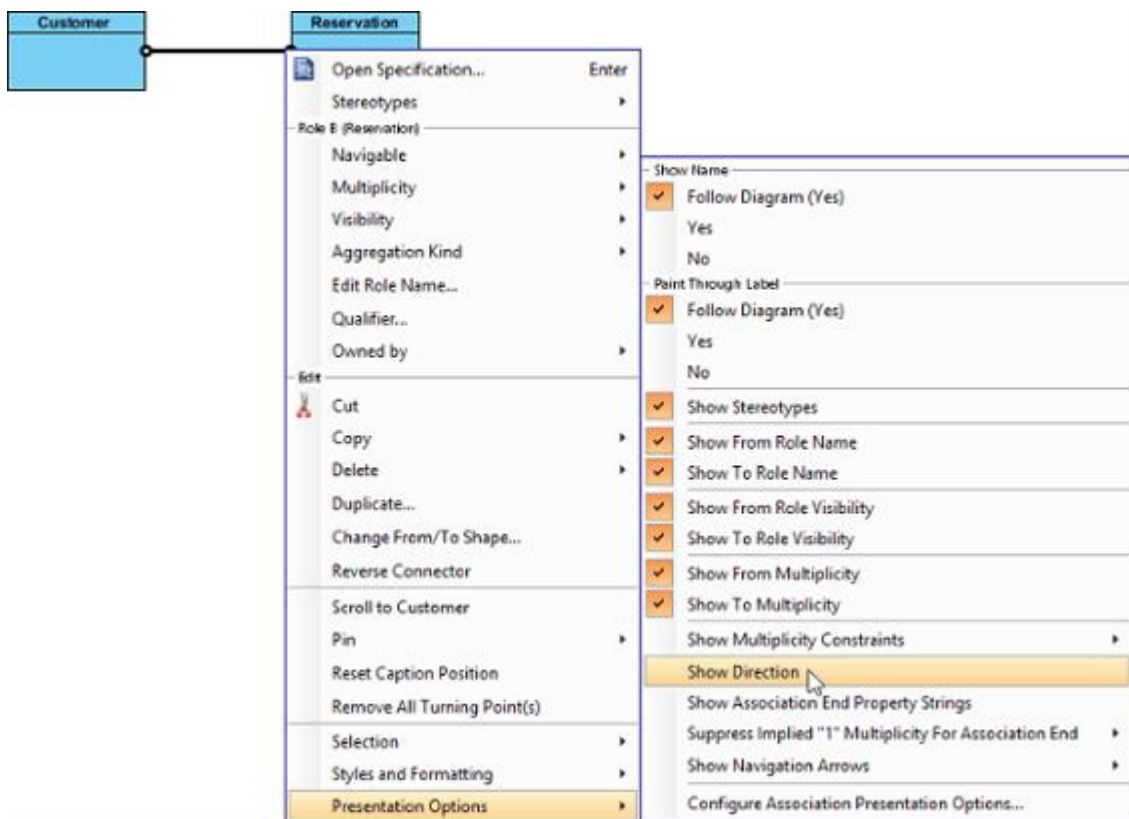


Рисунок 67 – Определение направления связи

Направление стрелки покажет связанный элемент диаграммы (Рисунок 20).



Рисунок 68 – Результат определения ассоциации

Для создания связи обобщения между классами выберите **Generalization -> Class** (Рисунок 23).

Перетащите курсор мыши на пустое место на диаграмме для создания нового или на уже существующий класс для соединения. Отпустите кнопку мыши (Рисунок 24).



Рисунок 69 – Выбор связи

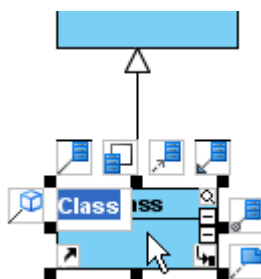


Рисунок 70 – Создание связи обобщения

Для создания атрибута щелкните правой кнопкой мыши на классе и выберите **Add > Attribute** в выпадающем меню (Рисунки 23-24).

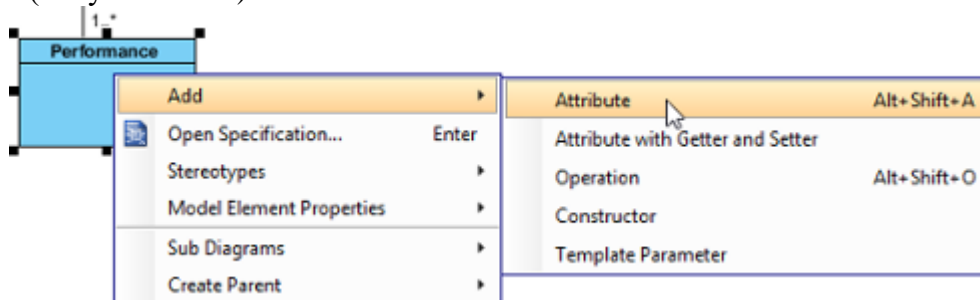


Рисунок 71 – Создание атрибута

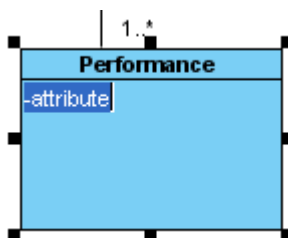


Рисунок 72 – Класс с созданным атрибутом

Для ускорения процесса создания атрибутов нужно сразу же после созданного атрибута нажать на клавиатуре клавишу **Enter**.

Для добавления типа данных атрибута выделите его и нажмите **Enter** либо щелкните правой кнопкой мыши и выберите **Open Specification...** в окне свойств выберите тип данных (рис. 25).

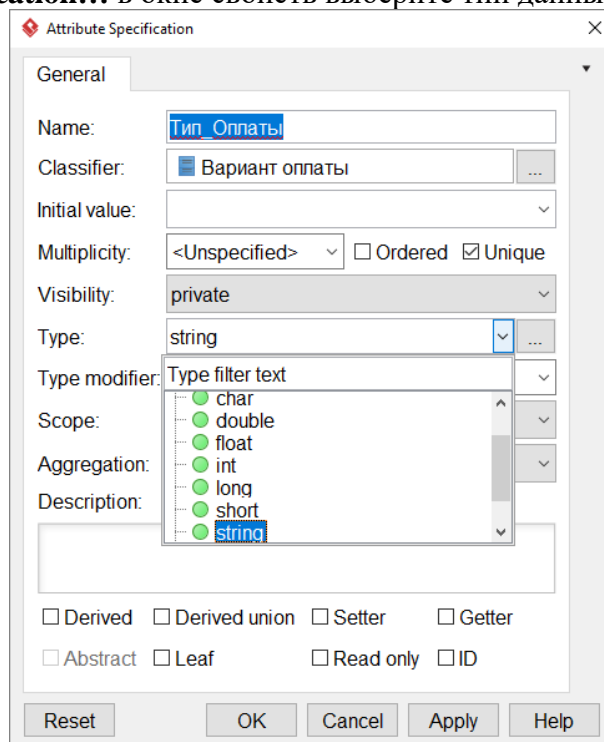


Рисунок 73 – Задание типа данных атрибута

Для создания операции щелкните правой кнопкой мыши на классе и выберите **Add > Operation** в выпадающем меню (Рисунки 26-27).

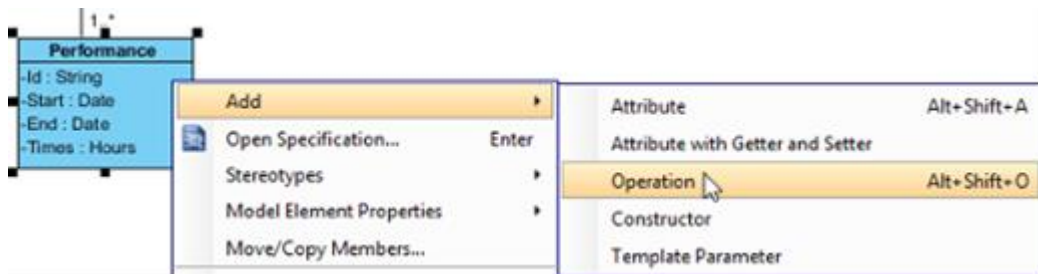


Рисунок 74 – Создание операции

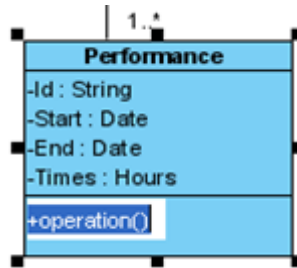


Рисунок 75 – Созданная операция

Для того чтобы переупорядочить члены класса необходимо зацепить курсором мышки один из членов и перетащить (Рисунки 28-29)

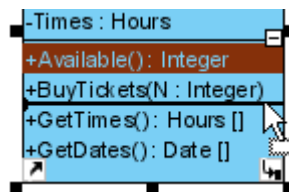


Рисунок 76 – Перетаскивание члена класса

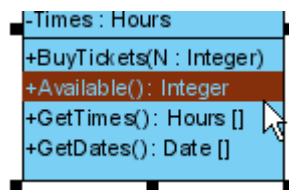


Рисунок 77 – Результат перетаскивания

Для копирования члена из одного класса в другой необходимо выбрать курсором мыши этот член класс с зажатой клавишей **Ctrl** и указать другой класс (Рисунки 30-31).

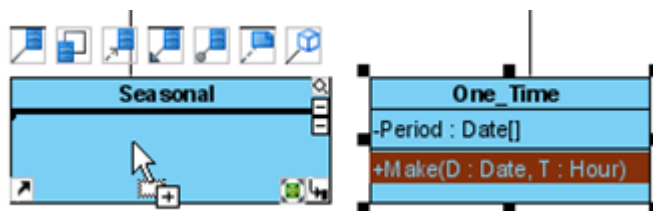


Рисунок 78 – Копирование члена класса

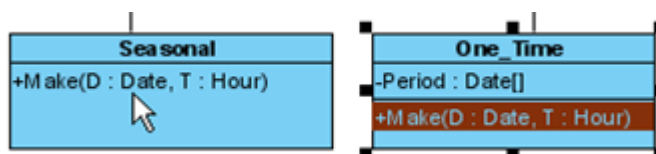


Рисунок 79 – Результат копирования члена класса

Перетаскивание члена класса из одного класса в другой производится подобным образом что и копирование, за исключением того, что не нужно зажимать клавишу **Ctrl**.

Для того чтобы выбрать все члены класса необходимо выбрать один член класса и нажать сочетание клавиш **Ctrl+A**.

Связывание классов производится по следующему алгоритму.

1. Выбрать тип связи из панели инструментов (Рисунок 32).

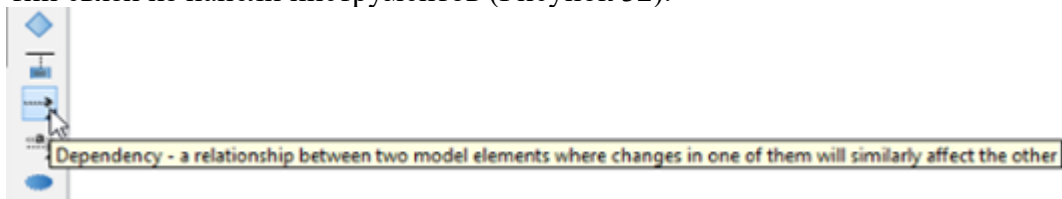


Рисунок 80 – Выбор зависимости

2. Переместить курсор мыши на атрибут-источник исходного класса (Рисунок 33).



Рисунок 81 – Определение связи по члену класса

3. Зажать клавишу мыши и не отпустить ее.

4. Переместить курсор мыши на член другого класса для связи (Рисунок 34).

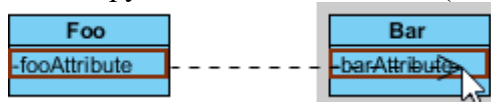


Рисунок 82 – Определение члена класса в целевом классе

5. Отпустить клавишу мыши для определения соединения. В спецификации можно убедиться, что связь определена именно по указанным атрибутам (Рисунок 35).



Рисунок 83 – Созданная связь

Основываясь, на описанных выше принципах построения диаграммы классов, постройте диаграмму классов для системы продажи товаров по каталогу в соответствии с рисунком 36.

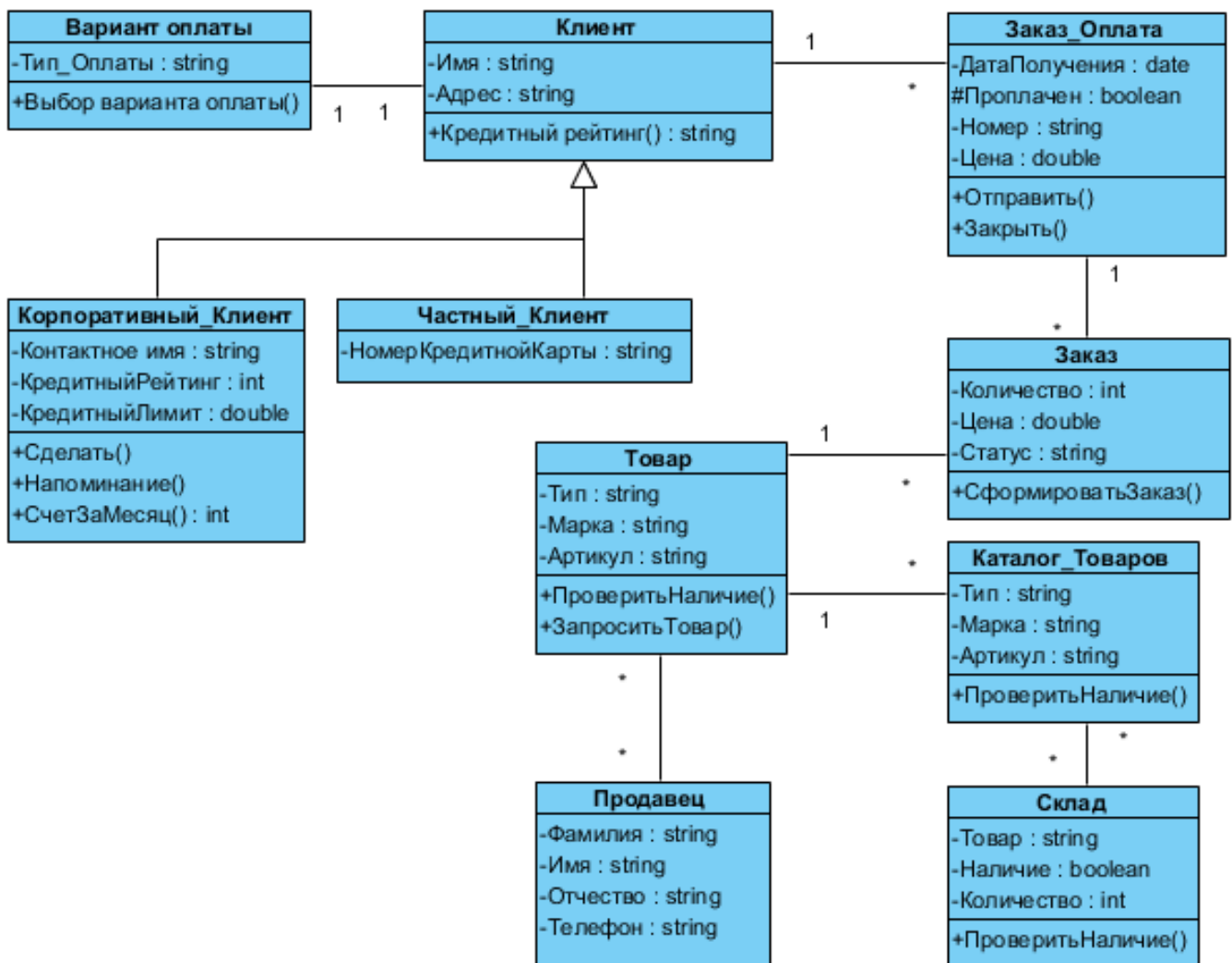


Рисунок 84 – Фрагмент диаграммы классов, описывающей реализацию систем продаж товаров по каталогу

4 Задание

Построить диаграмму классов в соответствии с вариантом.

Отчет по практическому занятию выполняется в формате MS Word, который содержит пошаговое описание процесса построения диаграммы, а также скриншоты результатов согласно заданию.

5 Варианты

1. «Отдел кадров»;
2. «Агентство аренды»;
3. «Аптека»;
4. «Ателье»;
5. «Аэропорт»;
6. «Библиотека»;
7. «Кинотеатр»;
8. «Поликлиника»;
9. «Автосалон»;
10. «Таксопарк».
11. «Издательство»;
12. «Прокат велосипедов»;
13. «Спортивный клуб».

6 Контрольные вопросы

1. Дайте определение понятию «диаграмма классов» и ее назначение.
2. Дайте определение следующим понятиям «класс», «атрибут», «операция», «отношение».

3. Опишите отношение ассоциаций между экземплярами классов.
4. Опишите отношение обобщения между экземплярами классов.
5. Опишите отношение агрегации между экземплярами классов.
6. Опишите отношение композиции между экземплярами классов.
7. Приведите пример графического представления основных компонентов диаграммы классов.
8. Дайте определение «квантор видимости» и его классификацию.

Практическая работа №20

Разработка диаграмм состояний с помощью Visual Paradigm for UML

Цель: изучение основ создания диаграмм состояний на языке UML с помощью case-средства Visual Paradigm for UML CE, применение приобретенных навыков для построения объектно-ориентированных моделей определенной предметной области.

1 Задачи

Основными задачами практической работы являются:

- получить навыки создания диаграмм состояний с помощью case-средства Visual Paradigm for UML CE;
- изучить основные стереотипы классов.

2 Краткие теоретические сведения

Для представления динамических особенностей взаимодействия элементов модели, в контексте реализации вариантов использования, предназначены диаграммы кооперации и последовательности. Однако для моделирования процессов функционирования большинства сложных систем, особенно систем реального времени, этих представлений недостаточно.

Диаграмма состояний (statechart diagram) – диаграмма, которая представляет конечный автомат, в которой вершины обозначают состояния, а дуги показывают переходы между двумя состояниями.

Назначение диаграммы состояний – описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение моделируемой системы в течение всего ее жизненного цикла. Диаграмма состояний представляет динамическое поведение сущностей, на основе спецификации их реакции на восприятие некоторых конкретных событий.

Диаграммы состояний используются для описания поведения отдельных систем и подсистем. Они также могут быть применены для спецификации функциональности экземпляров отдельных классов, т. е. для моделирования всех возможных изменений состояний конкретных объектов. Диаграмма состояний по существу является графом специального вида, который служит для представления конечного автомата.

Диаграммы состояний могут быть вложены друг в друга, образуя вложенные диаграммы для более детального представления состояний отдельных элементов модели. Для понимания семантики конкретной диаграммы состояний необходимо представлять особенности поведения моделируемой сущности, а также иметь общие сведения из теории конечных автоматов.

Конечный автомат (state machine) – модель для спецификации поведения объекта в форме последовательности его состояний, которые описывают реакцию объекта на внешние события, выполнение объектом действий, а также изменение его отдельных свойств.

В контексте языка UML понятие конечного автомата обладает дополнительной семантикой. Вершинами графа конечного автомата являются состояния и другие типы элементов модели, которые изображаются соответствующими графическими символами. Дуги графа служат для обозначения переходов из состояния в состояние. Конечный автомат описывает поведение отдельного объекта в форме последовательности состояний, охватывающих все этапы его жизненного цикла, начиная от создания объекта и заканчивая его уничтожением. Каждая диаграмма состояний представляет собой конечный автомат.

Состояние (state) – условие или ситуация в ходе жизненного цикла объекта, в течение которого он удовлетворяет логическому условию, выполняет определенную деятельность или ожидает события.

Состояние может быть задано в виде набора конкретных значений атрибутов объекта некоторого класса, при этом изменение отдельных значений этих атрибутов будет отражать изменение состояния моделируемого объекта или системы в целом. Однако не каждый атрибут класса может характеризовать состояние его объектов. Как правило, имеют значение только те свойства элементов системы, которые отражают динамический или функциональный аспект ее поведения. В этом случае состояние будет характеризоваться некоторым инвариантным условием, включающим в себя только принципиальные для поведения объекта или системы атрибуты классов и их значения.

Такое условие может соответствовать ситуации, когда моделируемый объект находится в состоянии ожидания возникновения внешнего события. В то же время нахождение объекта в некотором состоянии может быть связано с выполнением определенных действий. В последнем случае соответствующая

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 (рис. 3, а), из которого может только выходить стрелка-переход.



начальное состояние

конечное состояние

Рисунок 87 – Графическое изображение начального и конечного состояний

На самом верхнем уровне представления объекта переход из начального состояния может быть помечен событием создания (инициализации) данного объекта. В противном случае этот переход никак не помечается. Если этот переход не помечен, то он является первым переходом на диаграмме состояний в следующее за ним состояние. Каждая диаграмма или под-диаграмма состояний должна иметь единственное начальное состояние.

Конечное состояние (final state) – разновидность псевдосостояния, обозначающее прекращение процесса изменения состояний конечного автомата или нахождения моделируемого объекта в составном состоянии.

В этом состоянии должен находиться моделируемый объект или система по умолчанию после завершения работы конечного автомата. Оно служит для указания на диаграмме состояний графической области, в которой завершается процесс изменения состояний или жизненный цикл данного объекта.

Графически конечное состояние в языке UML обозначается в виде закрашенного кружка, помещенного в окружность (рис. 3, б), в которую может только входить стрелка-переход. Каждая диаграмма состояний или подсостояний может иметь несколько конечных состояний, при этом все они считаются эквивалентными на одном уровне вложенности состояний.

Переход (transition) – отношение между двумя состояниями, которое указывает на то, что объект в первом состоянии должен выполнить определенные действия и перейти во второе состояние.

Переход осуществляется при наступлении некоторого события: окончания выполнения деятельности (do activity), получении объектом сообщения или приемом сигнала. На переходе указывается имя события, а также действия, производимые объектом в ответ на внешние события при переходе из одного состояния в другое.

Событие (event) – спецификация существенных явлений в поведении системы, которые имеют местоположение во времени и пространстве.

Формально, событие представляет собой спецификацию факта, имеющего место в пространстве и во времени. Про события говорят, что они «происходят», при этом отдельные события должны быть упорядочены во времени. После наступления события нельзя уже вернуться к предыдущим, если такая возможность явно не предусмотрена в модели.

Семантика понятия события фиксирует внимание на внешних проявлениях качественных изменений, происходящих при переходе моделируемого объекта из состояния в состояние.

Переход называется **триггерным**, если его специфицирует событие-триггер, связанное с внешними условиями по отношению к рассматриваемому состоянию.

В этом случае рядом со стрелкой триггерного перехода обязательно указывается имя события в форме строки текста, начинающейся со строчной буквы. Наиболее часто в качестве имен триггерных переходов задают имена операций, вызываемых у тех или иных объектов системы. После имени такого события следуют круглые скобки для явного задания параметров соответствующей операции. Если таких параметров нет, то список параметров со скобками может отсутствовать. Например, переход на рис. 4, а,

является триггерным, поскольку с ним связано конкретное событие-триггер, происходящее асинхронно при срабатывании некоторого датчика.

Переход называется **нетриггерным**, если он происходит по завершении выполнения деятельности в данном состоянии.

Нетриггерные переходы часто называют переходами по завершении деятельности. Для них рядом со стрелкой перехода не указывается никакого имени события, а в исходном состоянии должна быть описана внутренняя деятельность, по окончании которой произойдет тот или иной нетриггерный переход.



Рисунок 88 – Графическое изображение триггерного (а) и нетриггерного (б) переходов на диаграмме состояний

Сторожевое условие (guard condition) – логическое условие, записанное в прямых скобках и представляющее собой булево выражение. При этом булево выражение должно принимать одно из двух взаимно исключающих значений: «истина» или «ложь». Из контекста диаграммы состояний должна явно следовать семантика этого выражения, а для записи выражения может использоваться обычный язык, псевдокод или язык программирования.



Рисунок 89 – Триггерные и нетриггерные переходы на диаграмме состояний

Изображенный фрагмент диаграммы состояний (рис. 5) моделирует изменение состояний банкомата при проверке ПИН-кода. Нетриггерные переходы на данной диаграмме помечены сторожевыми условиями, которые исключают конфликт между ними. Что касается триггерного перехода, помеченного событием отмена транзакции, то он происходит независимо от проверки ПИН-кода в том случае, когда клиент решил отказаться от ввода ПИН-кода.

Выражение действия (action expression) представляет собой вызов операции или передачу сообщения, имеет атомарный характер и выполняется сразу после срабатывания соответствующего перехода до начала действий в целевом состоянии. Выражение действия выполняется в том и только в том случае, когда переход срабатывает.

Атомарность действия означает, что оно не может быть прервано никаким другим действием до тех пор, пока не закончится его выполнение. Данное действие может оказывать влияние как на сам объект, так и на его окружение, если это с очевидностью следует из контекста модели. Данное выражение записывается после знака "/" в строке текста, присоединенной к соответствующему переходу.

В качестве примера выражения действия перехода (рис. 6) может служить отображение сообщения на экране банкомата в том случае, когда запрашиваемая клиентом сумма превосходит остаток на его счету. В случае если кредит не превышен, то происходит переход в состояние получения наличных.

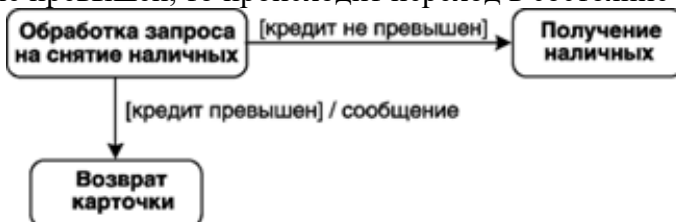


Рисунок 90 – Выражение действия перехода на диаграмме состояний

Составное состояние (composite state) – сложное состояние, которое состоит из других вложенных в него состояний (состояние-композит). Вложенные состояния выступают по отношению к составному состоянию как **подсостояния** (substate). И хотя между ними имеет место отношение композиции, графически все вершины диаграммы, которые соответствуют вложенным состояниям, изображаются внутри символа составного состояния (рис. 7). В этом случае размеры графического символа составного состояния увеличиваются, так чтобы вместить в себя все подсостояния.



Рисунок 91 – Графическое представление составного состояния с двумя вложенными в него последовательными подсостояниями

Составное состояние может содержать или несколько последовательных подсостояний, или несколько параллельных конечных подавтоматов. Каждое состояние-композит может уточняться только одним из указанных способов. При этом любое из подсостояний, в свою очередь, может быть состоянием-композитом и содержать внутри себя другие вложенные подсостояния. Количество уровней вложенности составных состояний в языке UML не фиксировано.

Последовательные подсостояния (sequential substates) – вложенные состояния состояния-композиата, в рамках которого в каждый момент времени объект может находиться в одном и только одном подсостоянии.

Поведение объекта в этом случае представляет собой последовательную смену подсостояний, от начального до конечного. Моделируемый объект или система продолжает находиться в составном состоянии, тем не менее, введение в рассмотрение последовательных подсостояний позволяет учесть более тонкие логические аспекты его внутреннего поведения.

В качестве примера моделируемой системы стоит рассмотреть обычный телефонный аппарат. Он может находиться в различных состояниях, в частности в состоянии дозвона до абонента. Очевидно, для того чтобы позвонить, необходимо снять телефонную трубку, услышать тоновый сигнал, после чего набрать нужный телефонный номер. Таким образом, состояние дозвона до абонента является составным и состоит из двух последовательных подсостояний: Телефонная трубка поднята и Набор телефонного номера. Фрагмент диаграммы состояний для этого примера содержит одно состояние-композит, которое состоит из двух последовательных подсостояний (рис. 8).



Рисунок 92 – Пример составного состояния с двумя вложенными последовательными подсостояниями

Некоторых пояснений могут потребовать переходы. Два из них специфицируют событие-триггер, которое имеет имя: набор цифры(n) с параметром n. В качестве параметра, как нетрудно предположить, выступает отдельная цифра на диске телефонного аппарата. Переход из начального подсостояния не содержит никакой строки текста. Последний переход в конечное подсостояние также не имеет события-триггера, но имеет сторожевое условие, проверяющее полноту набранного номера абонента. Только в

случае истинности этого условия телефонный аппарат может перейти в конечное состояние для состояния-композиата Дозвон до абонента.

Каждое составное состояние должно содержать в качестве вложенных состояний начальное и конечное состояния. При этом начальное подсостояние является исходным, когда происходит переход объекта в данное составное состояние. Если составное состояние содержит внутри себя конечное состояние, то переход в это вложенное конечное состояние означает завершение нахождения объекта в данном составном состоянии. Важно помнить, что для последовательных подсостояний начальное и конечное состояния должны быть единственными в каждом составном состоянии.

Это можно объяснить следующим образом. Каждая совокупность вложенных последовательных подсостояний представляет собой конечный подавтомат того конечного автомата, которому принадлежит рассматриваемое составное состояние. Поскольку каждый конечный автомат может иметь по определению единственное начальное и единственное конечное состояния, то для любого его конечного подавтомата это условие также должно выполняться.

Параллельные подсостояния (concurrent substates) – вложенные состояния, используемые для спецификации двух и более конечных подавтоматов, которые могут выполняться параллельно внутри составного состояния.

Каждый из конечных подавтоматов занимает некоторую графическую область внутри составного состояния, которая отделяется от остальных горизонтальной пунктирной линией. Если на диаграмме состояний имеется составное состояние с вложенными параллельными подсостояниями, то объект может одновременно находиться в каждом из этих подсостояний.

Отдельные параллельные подсостояния могут, в свою очередь, состоять из нескольких последовательных подсостояний (рис. 9). В этом случае по определению моделируемый объект может находиться только в одном из последовательных подсостояний каждого подавтомата. Таким образом, для фрагмента диаграммы состояний (рис. 9) допустимо одновременное нахождение объекта только в следующих подсостояниях: (А, В, Г), (Б, В, Г), (А, В, Д), (Б, В, Д).



Рисунок 93 – Графическое изображение состояния-композиата с вложенными параллельными подсостояниями

Несовместимое подсостояние (disjoint substate) – подсостояние, в котором подсистема не может находиться одновременно с другими подсостояниями одного и того же составного состояния.

В этом контексте недопустимо нахождение объекта одновременно в несовместимых подсостояниях (А, Б, В) или (В, Г, Д).

Может оказаться необходимым учесть ту часть деятельности, которая была выполнена на момент выхода из этого состояния-композиата, чтобы не начинать ее выполнение сначала. Для этой цели в языке UML существует историческое состояние.

Историческое состояние (history state) – псевдосостояние, используемое для запоминания того из последовательных подсостояний, которое было текущим в момент выхода из составного состояния.

Историческое состояние применяется только в контексте составного состояния. При этом существует две разновидности исторического состояния: неглубокое или недавнее и глубокое или давнее (рис. 10).



Рисунок 94 – Графическое изображение недавнего (а) и давнего (б) исторического состояния

Неглубокое историческое состояние (shallow history state) обозначается в форме небольшой окружности, в которую помещена латинская буква «Н» (рис. 10, а). Это состояние обладает следующей семантикой. Во-первых, оно является первым подсостоянием в составном состоянии, и переход извне в рассматриваемое составное состояние должен вести непосредственно в данное историческое состояние.

Во-вторых, при первом попадании в неглубокое историческое состояние оно не хранит никакой истории. Другими словами, при первом переходе в недавнее историческое состояние оно заменяет собой начальное состояние соответствующего конечного подавтомата.

Далее могут последовательно изменяться вложенные подсостояния. Если в некоторый момент происходит выход из составного состояния (например, в случае наступления некоторого события), то рассматриваемое историческое состояние запоминает то из подсостояний, которое было текущим на момент выхода из данного составного состояния. При последующем входе в это составное состояние неглубокое историческое подсостояние имеет непустую историю и сразу отправляет конечный подавтомат в запомненное подсостояние, минуя все предшествующие ему подсостояния.

Глубокое историческое состояние (deep history state или состояние глубокой истории) также обозначается в форме небольшой окружности, в которую помещена латинская буква «H» с дополнительным символом "*" (рис. 10, б), и служит для запоминания всех подсостояний любого уровня вложенности для исходного составного состояния.

В отдельных случаях возникает необходимость явно показать ситуацию, когда переход может иметь несколько исходных состояний или целевых состояний. Такой переход получил название – **параллельный переход**. Введение в рассмотрение параллельных переходов может быть обусловлено необходимостью синхронизировать и/или разделить отдельные процессы управления на параллельные нити без спецификации дополнительной синхронизации в параллельных конечных подавтоматах.

Графически такой переход изображается вертикальной черточкой, аналогично обозначению перехода в известном формализме сетей Петри. Если параллельный переход имеет две или более исходящих из него дуг (рис. 11, а), то его называют разделением (fork). Если же он имеет две или более входящие дуги (рис. 11, б), то его называют слиянием (join). Текстовая строка спецификации параллельного перехода записывается рядом с черточкой и относится ко всем входящим или исходящим дугам.



Рисунок 95 – Графическое изображение перехода-разделения в параллельные подсостояния (а) и перехода-слияния из параллельных подсостояний (б)

Состояние синхронизации (synch state) – псевдосостояние в конечном автомате, которое используется для синхронизации параллельных областей конечного автомата.

Синхронизирующее состояние обозначается небольшой окружностью, внутри которой помещен символ звездочки "*". Оно используется совместно с переходом-слиянием или переходом-разделением для того, чтобы явно указать события в других конечных подавтоматах, оказывающие непосредственное влияние на поведение данного подавтомата.

Так, например, при включении компьютера с некоторой сетевой операционной системой параллельно начинается выполнение нескольких процессов. В частности, происходит проверка пароля пользователя и запуск различных служб. При этом работа пользователя на компьютере станет возможной только в случае успешной его аутентификации, в противном случае компьютер может быть выключен.

Рассмотренные особенности синхронизации этих параллельных процессов учтены на соответствующей диаграмме состояний с помощью синхронизирующего состояния (рис. 12).



Рисунок 96 – Диаграмма состояний для примера включения компьютера

Пример: Программное средство представляет собой базу данных и обеспечивает корректировку данных, а именно в БД «Группы» может изменяться перечень предметов в соответствии с курсом группы и отделением; в БД «Предметы» могут изменяться номера аудиторий и фамилии преподавателей; осуществляет поиск по ФИО преподавателя, номеру аудитории, названию предмета и номеру группы. Программное средство составляет расписание работы для конкретного преподавателя на неделю; для группы на неделю; для группы по конкретному предмету, а также отчет о загрузке аудиторий на каждый день.



Рисунок 97 – Пример диаграммы состояний

3 Методика выполнения

В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

1. Запустите Visual Paradigm for UML SE.
2. Откройте проект, содержащий диаграмму вариантов использования, построенную в практическом занятии №13.
3. Для добавления новой диаграммы достаточно в меню Diagram > New Diagram выбрать тип диаграммы State Machine Diagram и нажмите кнопку Next. Заданн имя диаграммы и нажать OK.
4. Либо, в ранее созданной диаграмме классов, щелкнуть правой кнопкой мыши по классу, для которого необходимо создать диаграмму состояний и в контекстном меню выбрать **Sub Diagrams > 1) New Diagram > State Machine Diagram** > при необходимости изменить имя диаграммы и нажать OK.
- 2) **Existing Diagram** > выбрать в списке ранее созданную диаграмму и добавить ее в качестве поддиаграммы.
5. В результате отобразится рабочая область с элементами для построения диаграммы состояний (рис. 14).

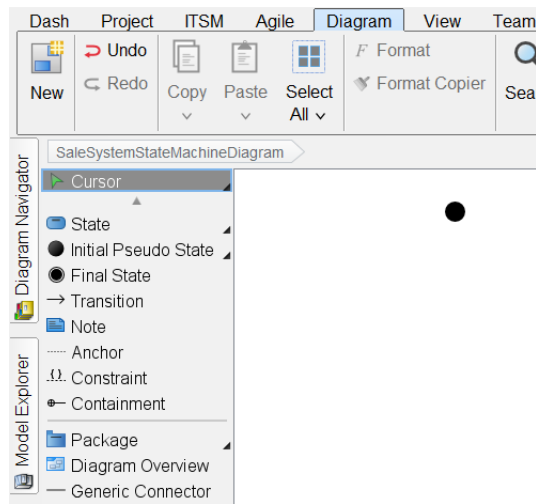


Рисунок 98 – Рабочая область для построения диаграммы состояний

Далее будут описаны принципы построения диаграммы состояний.

Для создания состояния необходимо на панели инструментов выбрать State и затем курсором мыши выбрать место расположения на диаграмме (Рисунок 15). После создания класса необходимо задать его имя (Рисунок 16).

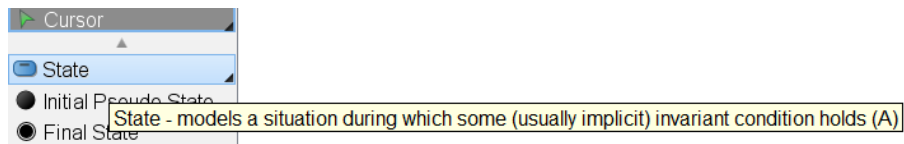


Рисунок 99 – Создание состояния

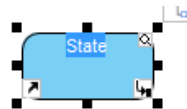


Рисунок 100 – Задание имени в созданном состоянии

Для создания перехода между начальным состоянием и последующим состоянием или между состояниями выберите **Transition > State** (Рисунок 17).

Перетащите курсор мыши на пустое место на диаграмме для создания нового или на уже существующее состояние для соединения. Отпустите кнопку мыши (Рисунок 17).

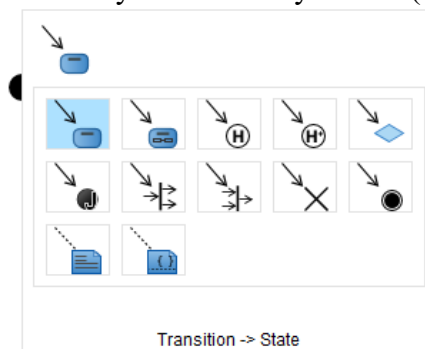


Рисунок 101 – Выбор типа перехода

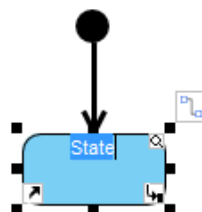


Рисунок 102 – Создание перехода

Для добавления внутренних действий, нажмите правой кнопкой мыши по состоянию и выберите пункт **Open Specification...** добавить действия можно посредством определения значений полей **Entry**, **Do**, **Exit Activity** (Рисунок 19). Либо выделите состояние и нажмите **Enter** (рисунок 19).

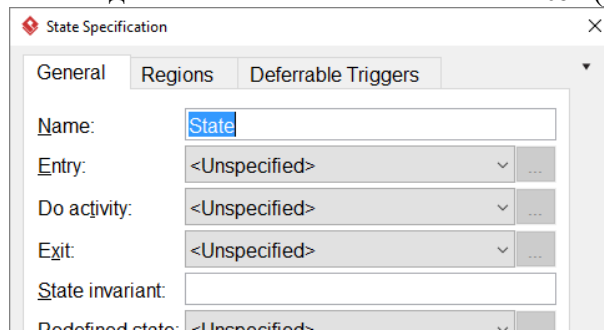


Рисунок 103 – Изменение внутренних действий

Например, для добавления внутреннего действия **Entry** щелкните раскрывающийся список и выберите **Create Activity...** (рисунок 20).

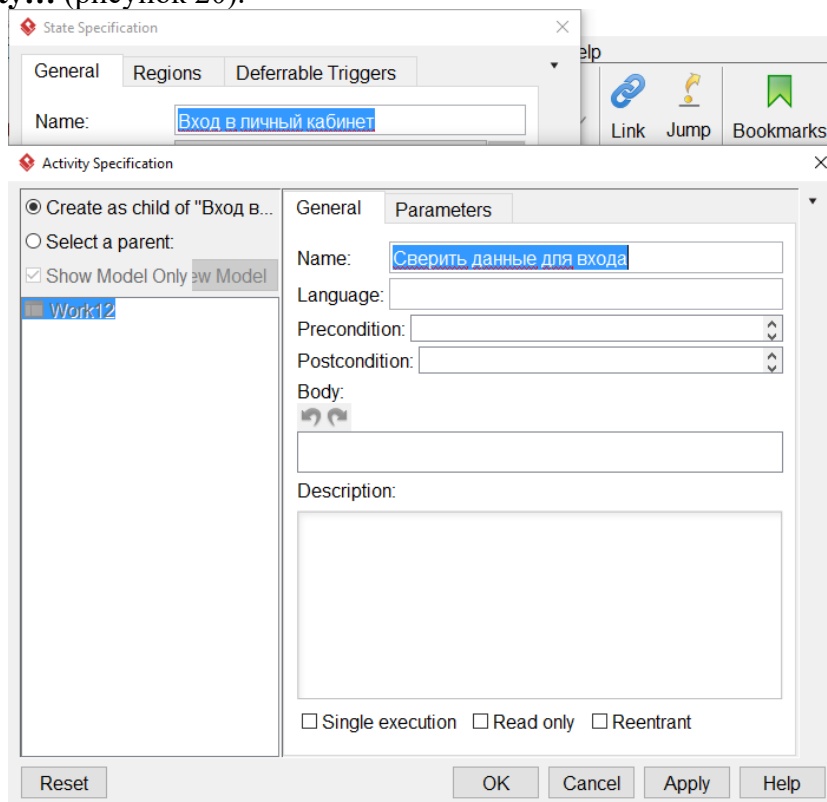


Рисунок 104 – Создание внутреннего действия

Для добавления описания перехода дважды щелкните по нему и введите описание (рисунок 21).

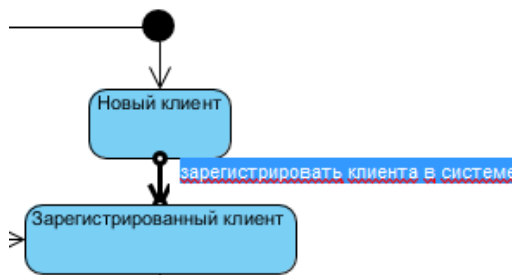


Рисунок 105 – Добавление описания перехода

Чтобы добавить переход разветвления или объединения наведите курсор в правый нижний угол элемента **Initial Pseudo State** и выберите **Fork** или **Join** (рисунок 22). После добавления элемента, для того чтобы изменить ориентацию на горизонтальную, щелкните правой кнопкой мыши по элементу и в контекстном меню выберите **Orientation > Horizontal**.

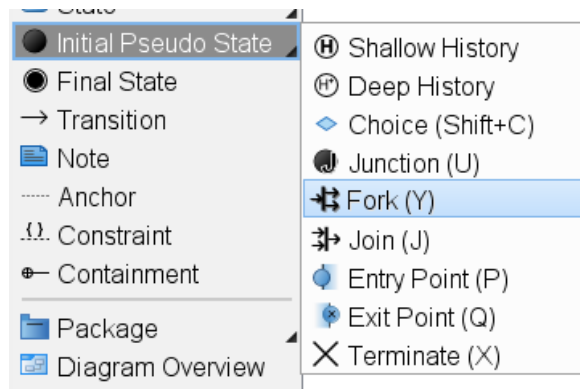


Рисунок 106 – Добавление перехода разветвления

Для того чтобы вернуться к диаграмме классов, нажмите на имя класса в иерархии диаграмм (рисунок 22).

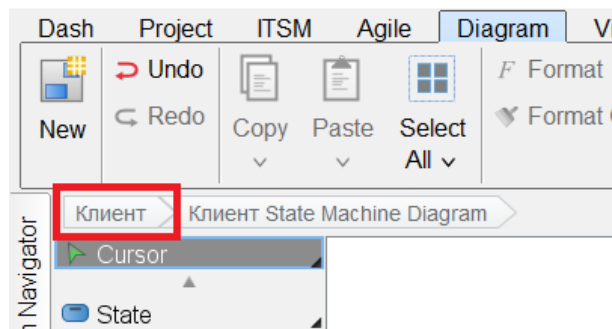


Рисунок 107 – Открытие диаграммы классов

Основываясь, на описанных выше принципах построения, постройте диаграммы состояний для системы продажи товаров по каталогу в соответствии с рисунками 24-27.

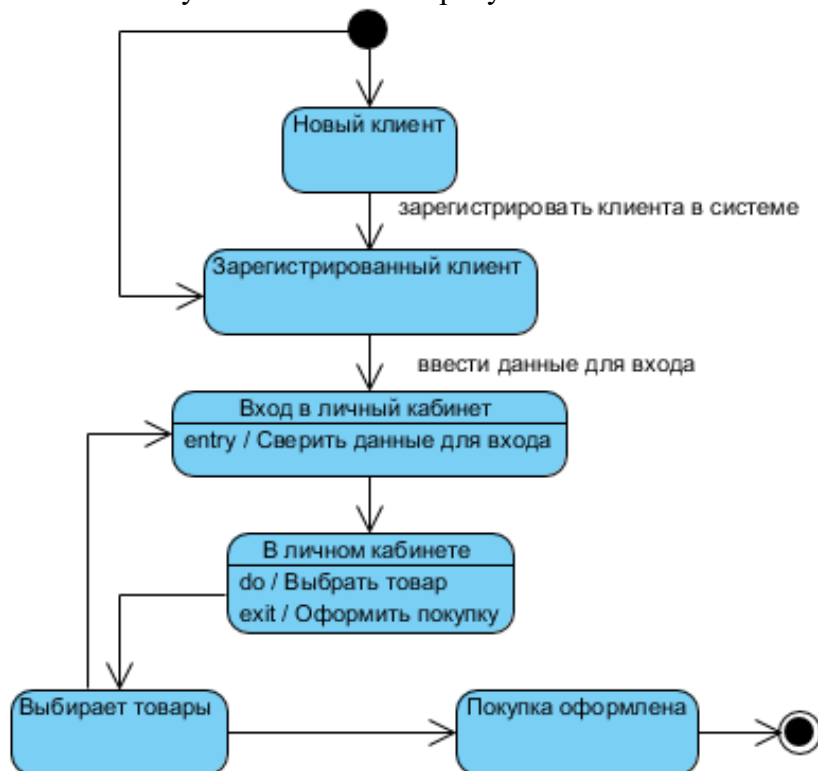


Рисунок 108 – Диаграмма состояний для класса «Клиент»

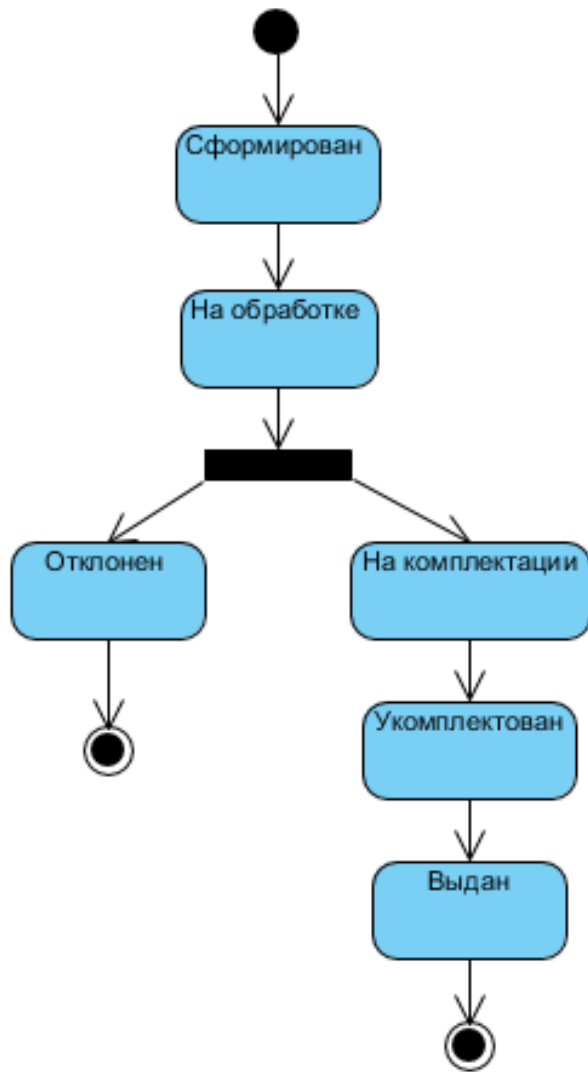


Рисунок 109 – Диаграмма состояний для класса «Заказ»

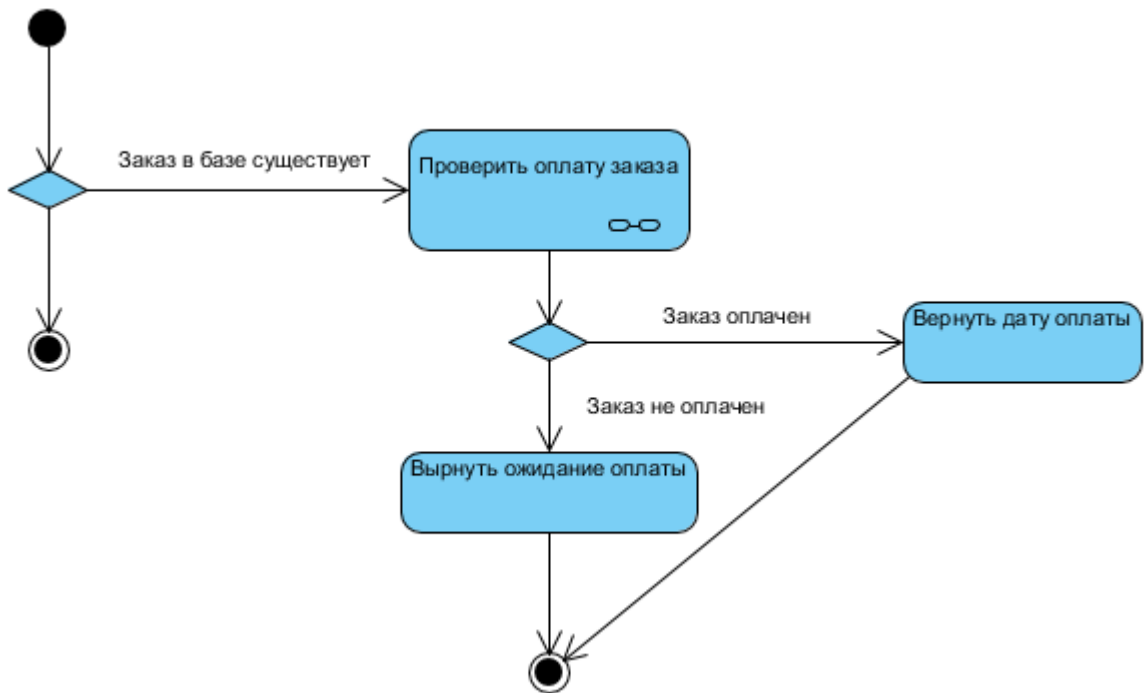


Рисунок 110 – Диаграмма состояний для класса «Заказ_Оплата»

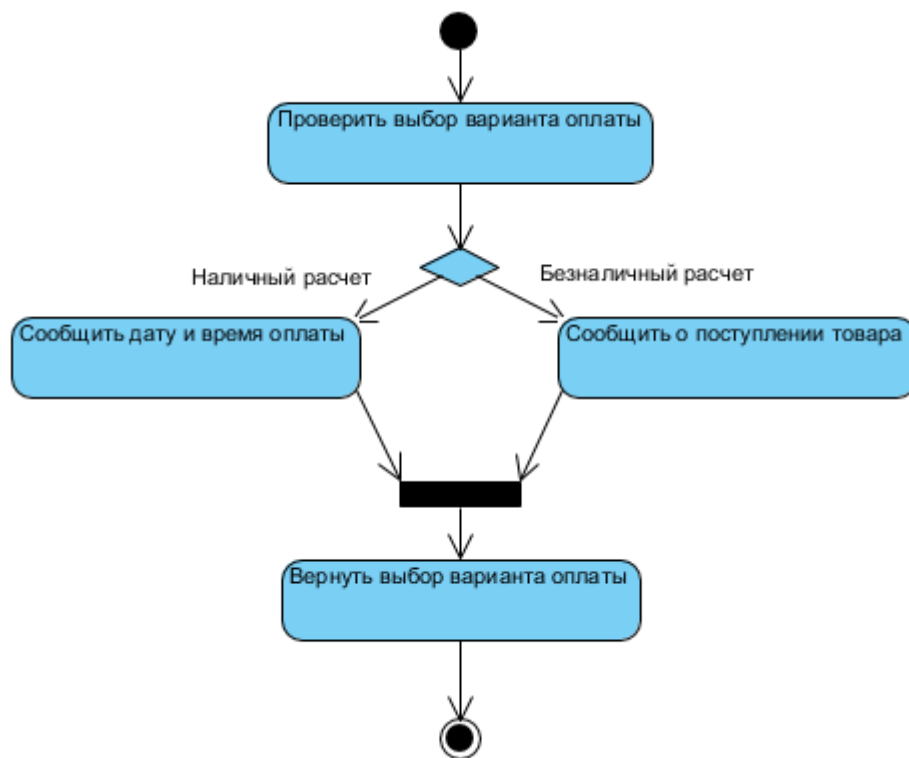


Рисунок 111 – Диаграмма состояний для класса «Вариант_Оплаты»

4 Задание

Построить диаграммы состояний для всех классов диаграммы классов, построенной в практическом занятии №19 в соответствии с вариантом.

Отчет по практическому занятию выполняется в формате MS Word, который содержит пошаговое описание процесса построения диаграммы, а также скриншоты результатов согласно заданию.

5 Варианты

1. «Отдел кадров»;
2. «Агентство аренды»;
3. «Аптека»;
4. «Ателье»;
5. «Аэропорт»;
6. «Библиотека»;
7. «Кинотеатр»;
8. «Поликлиника»;
9. «Автосалон»;
10. «Таксопарк».
11. «Издательство»;
12. «Прокат велосипедов»;
13. «Спортивный клуб».

6 Контрольные вопросы

1. Дайте определение понятию «диаграмма состояний».
2. Опишите назначение диаграммы состояний.
3. Дайте определение понятию «конечный автомат».
4. Дайте определение понятиям «состояния», «действие», «псевдосостояние». Графическое изображение состояния.
5. Что представляет собой переход? Какие бывают переходы, в чем их различие?
6. Дайте определение понятию «событие».
7. Дайте определения следующим понятиям: «составное состояние», «последовательные подсостояния», «параллельные подсостояния», «несовместимое подсостояние», «историческое состояние», «параллельный переход», «состояние синхронизации».

Практическая работа №21

Разработка диаграмм деятельности с помощью Visual Paradigm for UML

Цель: изучение основ создания диаграмм деятельности на языке UML с помощью case-средства Visual Paradigm for UML CE, применение приобретенных навыков для построения объектно-ориентированных моделей определенной предметной области.

1 Задачи

Основными задачами практической работы являются:

– получить навыки создания диаграмм деятельности с помощью case-средства Visual Paradigm for UML CE.

2 Краткие теоретические сведения

При моделировании поведения проектируемой или анализируемой программной системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и процедурной реализации выполняемых системой операций. Для этой цели, как правило, используются блок-схемы или структурные схемы алгоритмов. Каждая такая схема акцентирует внимание на последовательности выполнения определенных процедур или элементарных операций, которые в совокупности приводят к получению желаемого результата. С увеличением сложности системы строгое соблюдение определенной последовательности выполняемых действий приобретает большое значение.

Для моделирования процесса выполнения операций в языке UML используются диаграммы деятельности. Применяемая в них графическая нотация во многом похожа на нотацию диаграммы состояний, поскольку на диаграммах деятельности также присутствуют обозначения состояний и переходов. Отличие заключается в семантике состояний, которые используются для представления деятельности и действий, а также в отсутствии на переходах сигнатуры событий. Каждое состояние на диаграмме деятельности соответствует выполнению некой операции, а переход в следующее состояние происходит только после завершения выполнения этой операции. Диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия или деятельности, а дугами – переходы от одного состояния действия к другому.

Диаграммы деятельности (activity diagram) – частный случай диаграмм состояний. Они позволяют реализовать в языке UML особенности процедурного и синхронного управления, обусловленного завершением внутренних действий и деятельности. Основным направлением использования диаграмм деятельности является визуализация особенностей реализации операций классов, когда необходимо представить алгоритмы их выполнения. При этом каждое состояние может являться выполнением операции определенного класса либо ее части, позволяя использовать диаграммы деятельности для описания реакций на внутренние события системы.

В контексте языка UML **деятельность** представляет собой совокупность отдельных вычислений, выполняемых автоматом. При этом отдельные элементарные вычисления могут приводить к результату или действию. На диаграмме деятельности отображается логика или последовательность перехода от одной деятельности к другой, при этом внимание фиксируется на результате деятельности. Сам же результат может привести к изменению состояния системы или возвращению некоторого значения. Диаграмма деятельности предназначена для моделирования поведения систем, хотя время в явном виде отсутствует на этой диаграмме. Ситуация здесь во многом аналогична диаграмме состояний, однако имеет ряд особенностей.

Состояние деятельности (activity state) – состояние в графе деятельности, которое служит для представления процедурной последовательности действий, требующих определенного времени. Переход из состояния деятельности происходит после выполнения специфицированной в нем ду-деятельности, при этом ключевое слово do в имени деятельности не указывается. Состояние деятельности не может иметь внутренних переходов, поскольку оно является элементарным.

Состояния деятельности могут быть подвергнуты дальнейшей декомпозиции, вследствие чего выполняемую деятельность можно представить с помощью других диаграмм деятельности. Состояния деятельности не являются атомарными, то есть могут быть прерваны. Предполагается, что для их завершения требуется заметное время.

Состояние деятельности можно представлять себе как составное состояние, поток управления которого включает только другие состояния деятельности и действий.



Рисунок 112 – Графическое изображение состояний деятельности действия

Состояние действия (action state) – специальный случай состояния с некоторым входным действием и, по крайней мере, одним выходящим из состояния переходом. Переход из состояния действия происходит после завершения входного действия. Состояние действия не может иметь внутренних переходов, поскольку оно является элементарным. Использование состояния действия заключается в моделировании шага выполнения алгоритма. Графически состояния деятельности и действия изображаются одинаковой фигурой, напоминающей прямоугольник, боковые стороны которого заменены выпуклыми дугами. Внутри этой фигуры записывается имя состояния деятельности или действия в форме выражения, которое должно быть уникальным в пределах одной диаграммы деятельности.

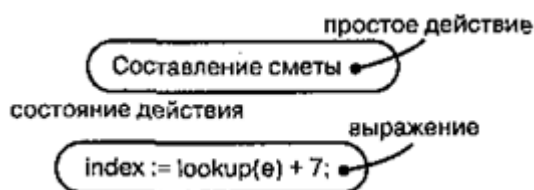


Рисунок 113 – Графическое изображение состояний действия на диаграмме деятельности

Действие может быть записано на естественном языке, псевдокоде или языке программирования. Никаких дополнительных или неявных ограничений при записи действий не накладывается. Рекомендуется в качестве имени простого действия использовать глагол с пояснительными словами (рис. 2). Если же действие может быть представлено в формальном виде, то оно записывается на том языке программирования, на котором предполагается реализовывать разрабатываемый проект.

Когда возникает необходимость представить на диаграмме деятельности сложное действие, состоящее из нескольких более простых, то используют специальное обозначение так называемого состояния под-деятельности. **Состояние под-деятельности** (subactivity state) – состояние в графе деятельности, которое служит для представления неатомарной последовательности шагов процесса. Это состояние является графом деятельности и обозначается специальной пиктограммой в правом нижнем углу символа состояния действия (рис. 3). Данная конструкция может применяться к любому элементу языка UML, который поддерживает «вложенность» своей структуры. При этом пиктограмма может быть дополнительно помечена типом вложенной структуры.



Рисунок 114 – Графическое изображение состояния под-деятельности

Каждая диаграмма деятельности должна иметь единственное начальное и конечное состояния. Они имеют такие же обозначения, как и на диаграмме состояний. При этом каждая деятельность начинается в начальном состоянии и заканчивается в конечном состоянии.

Саму диаграмму деятельности принято располагать таким образом, чтобы действия следовали сверху вниз или слева направо. В этом случае начальное состояние будет изображаться в верхней или левой части диаграммы, а конечное – в ее нижней или правой части. В интересах удобства визуального представления на диаграмме деятельности допускается изображать несколько конечных состояний. В этом случае все их принято считать эквивалентными друг другу.

Переход на диаграмме деятельности аналогичен переходу на диаграмме состояний. При построении диаграммы деятельности используются только нетриггерные переходы, т.е. такие, которые происходят

сразу после завершения деятельности или выполнения соответствующего действия. Такой переход передает управление в последующее состояние сразу, как только закончится действие или деятельность в предыдущем состоянии. На диаграмме такой переход изображается сплошной линией со стрелкой.

Если из состояния действия выходит единственный переход, то его можно никак не пометать. Если же таких переходов несколько, то при моделировании последовательной деятельности запускается только один из них. В этом случае для каждого из таких переходов должно быть явно записано собственное сторожевое условие в прямых скобках.

При этом для всех выходящих из некоторого состояния деятельности переходов должно выполняться требование истинности только одного из них. Подобный случай встречается тогда, когда последовательно выполняемая деятельность должна разделиться на альтернативные ветви в зависимости от значения промежуточного результата. Такая ситуация получила название **ветвления**, а для ее обозначения применяется специальный символ решения.

Графически ветвление на диаграмме деятельности обозначается символом решения (decision), изображаемого в форме небольшого ромба, внутри которого нет никакого текста (рис. 4). В ромб может входить только одна стрелка от того состояния действия, после выполнения, которого поток управления должен быть продолжен по одной из взаимно исключающих ветвей. Принято входящую стрелку присоединять к верхней или левой вершине символа решения. Выходящих стрелок может быть две или более. Для каждой из них явно указывается соответствующее сторожевое условие в форме булевского выражения.

Для графического объединения альтернативных ветвей на диаграмме деятельности рекомендуется использовать аналогичный символ в форме ромба, который в этом случае называют **соединением** (merge). Наличие этого символа, внутри которого также не записывается никакого текста, упрощает визуальный контроль логики выполнения процедурных действий на диаграмме деятельности (рис. 4 внизу). Входящих стрелок у символа соединения может быть несколько, они исходят от состояний действия, принадлежащих к одной из взаимно исключающих ветвей. Выходить из ромба соединения может только одна стрелка, при этом ни входящие, ни выходящая стрелки не должны содержать сторожевых условий. Исключением является ситуация, когда с целью сокращения диаграммы объединяют символ решения с символом соединения.

Пример: моделируется ситуация, возникающая в супермаркетах при оплате товаров. Как правило, заплатить за покупки можно либо наличными, либо по кредитной карточке. Если покупателем выбран вариант оплаты по кредитной карточке, то проверяется сумма баланса предъявленной к оплате кредитной карточки. При этом оплата происходит только в том случае, если общая стоимость приобретаемых товаров не превышает суммы баланса этой карточки. В противном случае оплаты не происходит, и товар остается у продавца.

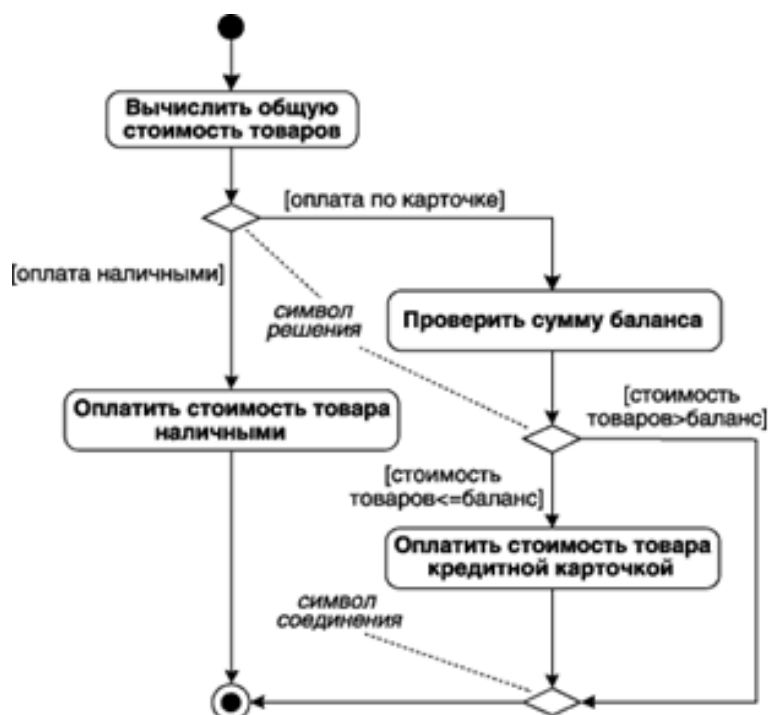


Рисунок 115 – Различные варианты ветвлений на диаграмме деятельности

На диаграмме деятельности изображаются параллельные процессы, поскольку распараллеливание вычислений существенно повышает быстродействие программных систем. В диаграммах деятельности для изображения параллельных процессов используется специальный символ.

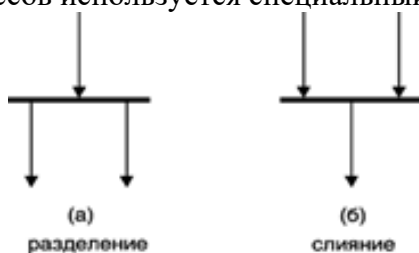


Рисунок 116 – Графическое изображение разделения и слияния параллельных потоков управления на диаграмме деятельности

На диаграммах деятельности такая черточка изображается отрезком горизонтальной, реже – вертикальной, линии, толщина которой несколько шире линий простых переходов диаграммы деятельности. При этом **разделение** (fork) имеет один входящий переход и несколько выходящих (рис. 6, а), которые изображаются отрезками вертикальных, реже – горизонтальных, линий. **Слияние** (join), наоборот, имеет несколько входящих переходов и один выходящий (рис. 6, б). Параллельные переходы на диаграмме деятельности можно изображать в удлиненной форме, а входящие и выходящие переходы вертикальными стрелками.

Пример: Регистрация пассажиров в аэропорту. Первоначально выполняется деятельность по проверке билета: если билет не действителен, он возвращается пассажиру; если билет действителен, то пассажиру выдается посадочный талон, в дополнение проверяется гражданство и наличие багажа у пассажира. Если есть багаж, то его проверка может быть выполнена параллельно, после чего пассажиру выдается талон на багаж. Если пассажир является иностранным гражданином, то дополнительно проверяется наличие у него визы. Если виза действительна, то проверка завершается успешно, и пассажир может проследовать на посадку. Если же виза не действительна, то для этого пассажира посадка оказывается невозможной, и ему не выдается посадочный талон и талон на багаж. Происходит прекращение всех выполняемых сотрудниками аэропорта действий.

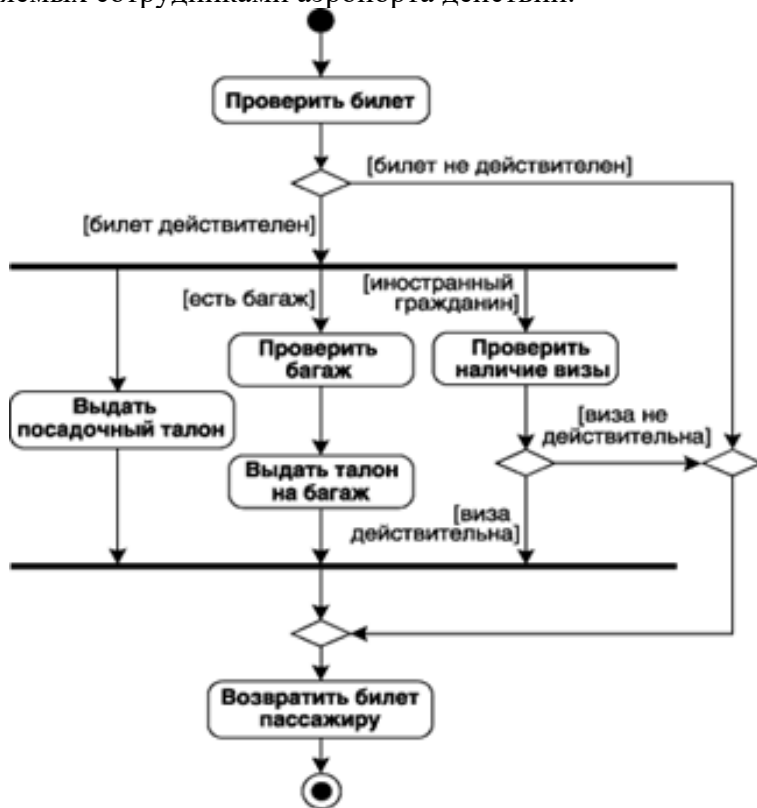


Рисунок 117 – Диаграмма деятельности для примера регистрации пассажиров в аэропорту

Дорожка (swimlane) – графическая область диаграммы деятельности, содержащая элементы модели, ответственность за выполнение которых принадлежит отдельным подсистемам.

В данном случае имеется в виду визуальная аналогия с плавательными дорожками в бассейне, если смотреть на соответствующую диаграмму деятельности сверху. При этом все состояния на диаграмме деятельности делятся на группы, разграниченные вертикальными линиями.

Две соседних линии и образуют дорожку, а группа состояний между этими линиями выполняется организационным подразделением (отделом, группой, отделением, филиалом) или сотрудником компании (рис. 7). В последнем случае принято указывать должность сотрудника, ответственного за выполнение определенных действий.

Названия подразделений или должностей явно указываются в верхней части дорожки. Пересекать линию дорожки могут только переходы, которые в этом случае обозначают выход или вход потока управления в соответствующее подразделение компании. Порядок следования дорожек не несет какой-либо семантической информации и определяется соображениями удобства.



Рисунок 118 – Вариант диаграммы деятельности с дорожками

Пример: фрагмент диаграммы деятельности торговой компании, обслуживающей клиентов в форме заказов. Подразделениями компании обычно являются отдел приема и оформления заказов, отдел продаж и склад. Этим подразделениям будут соответствовать три дорожки на диаграмме деятельности, каждая из которых специфицирует зону ответственности подразделения. В этом случае диаграмма деятельности включает в себе не только информацию о последовательности выполнения рабочих действий, но и о том, какое подразделение торговой компании должно выполнять то или иное действие (рис. 8).

После принятия заказа от клиента отделом приема и оформления заказов осуществляется распараллеливание деятельности на два потока (переход-разделение). Первый из них остается в этом же отделе и связан с получением оплаты от клиента за заказанный товар. Второй инициирует выполнение действия по регистрации заказа в отделе продаж (модель товара, размеры, цвет, год выпуска и пр.). Однако выдача товара со склада начинается только после того, как будет получена от клиента оплата за товар (переход-слияние). Затем выполняется подготовка товара к отправке и его отправка клиенту в отделе продаж. После завершения этих деятельностью заказ закрывается в отделе приема и оформления заказов.

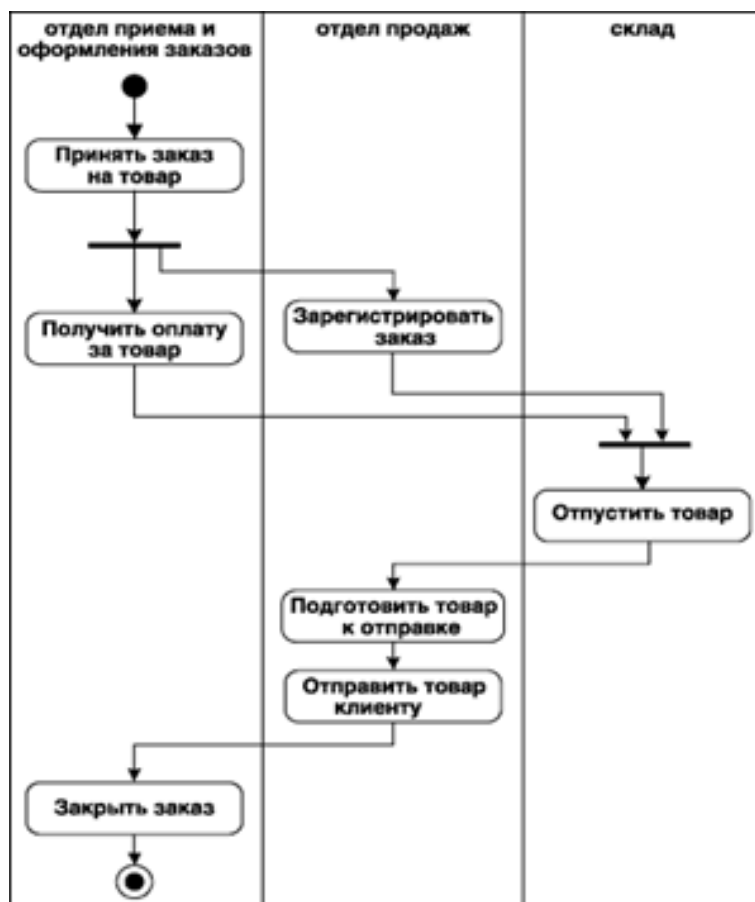


Рисунок 119 – Фрагмент диаграммы деятельности для торговой компании

Действия на диаграмме деятельности могут производиться над теми или иными объектами. Эти объекты либо инициируют выполнение действий, либо определяют результат этих действий. При этом действия специфицируют вызовы, которые передаются от одного объекта графа деятельности другому. Поскольку в таком ракурсе объекты играют определенную роль в понимании процесса деятельности, иногда возникает необходимость явно указать их на диаграмме деятельности.

На диаграмме деятельности с дорожками расположение объекта может иметь дополнительный смысл. А именно, если объект расположен на границе двух дорожек, то это может означать, что переход к следующему состоянию действия в соседней дорожке ассоциирован с нахождением документа в некотором состоянии. Если же объект расположен внутри дорожки, то и состояние этого объекта целиком определяется действиями данной дорожки.

В примере с торговой компанией центральным объектом процесса продажи является заказ или вернее состояние его выполнения. Вначале до обращения клиента заказ как объект отсутствует и возникает после контакта с клиентом. В результате фиксируется полученный заказ, потом он регистрируется в отделе продаж, затем он передается на склад, где после получения оплаты за товар оформляется окончательно. После того, как товар отправлен клиенту, эта информация вносится в заказ, и он считается выполненным (рис. 9).

Достоинством диаграммы деятельности является возможность визуализировать отдельные аспекты поведения рассматриваемой системы или реализации отдельных операций классов в виде процедурной последовательности действий. Таким образом, полная модель системы может содержать одну или несколько диаграмм деятельности, каждая из которых описывает последовательность реализации либо наиболее важных вариантов использования (типичный ход событий и все исключения), либо нетривиальных операций классов.

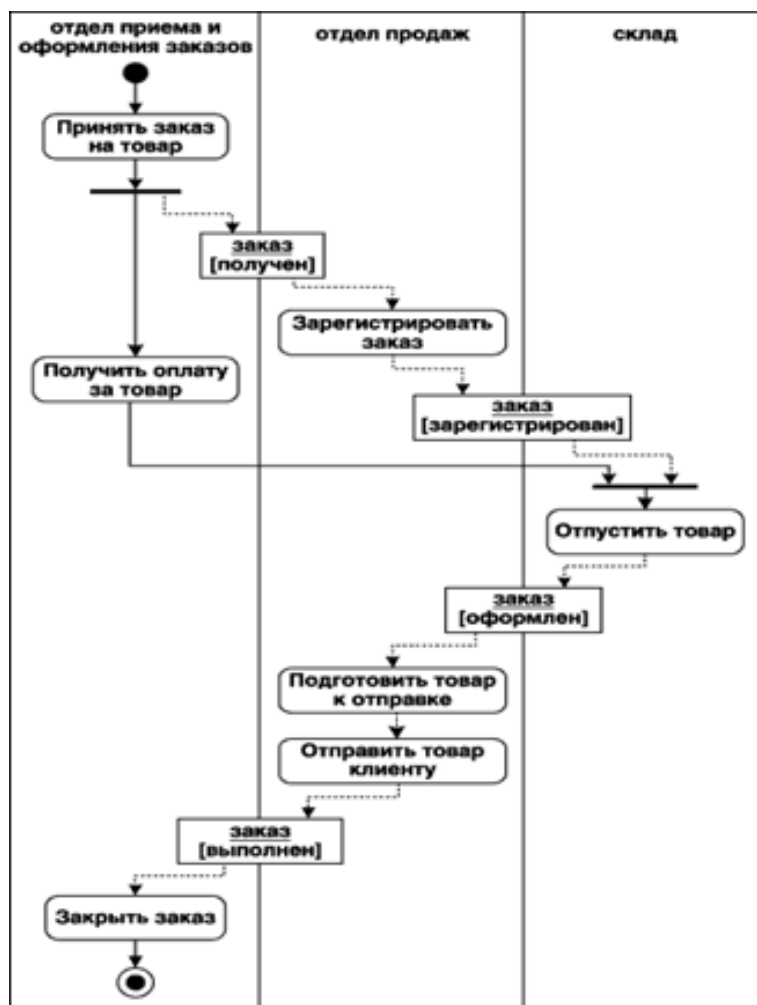


Рисунок 120 – Фрагмент диаграммы деятельности торговой компании с объектом-заказом

Пример: Программное средство представляет собой базу данных «Автоматизация процесса составления расписания в учебном заведении». Программное средство обеспечивает корректировку данных, а именно в БД «Группы» может изменяться перечень предметов в соответствии с курсом группы и отделением; в БД «Предметы» могут изменяться номера аудиторий и фамилии преподавателей; осуществляет поиск по ФИО преподавателя, номеру аудитории, названию предмета и номеру группы. Программное средство составляет расписание работы для конкретного преподавателя на неделю; для группы на неделю; для группы по конкретному предмету, а также отчет о загрузке аудиторий на каждый день.

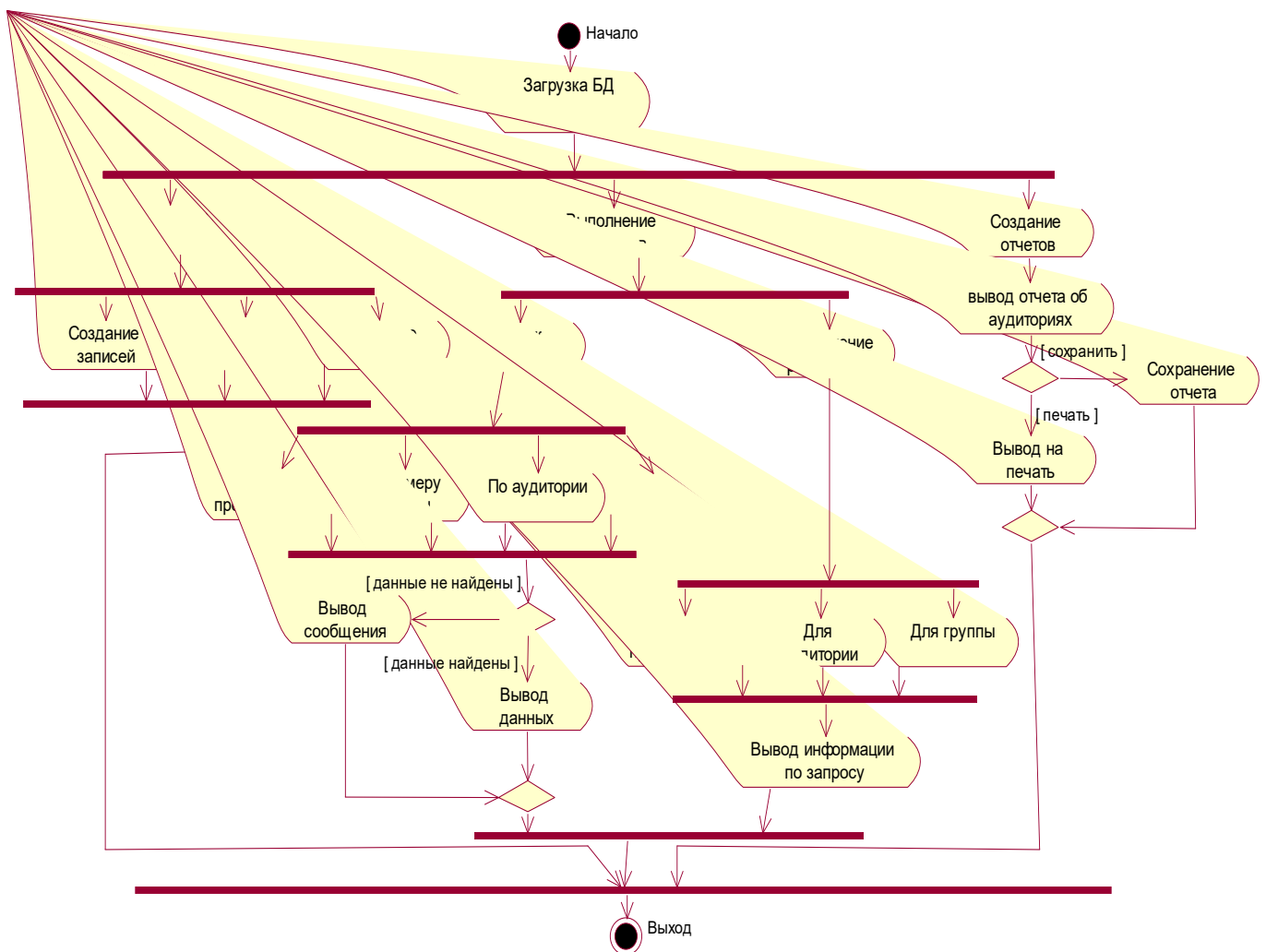


Рисунок 121 – Пример диаграммы деятельности

3 Методика выполнения

В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

1. Запустите Visual Paradigm for UML CE.
2. Откройте проект, содержащий диаграмму вариантов использования, построенную в практическом занятии №14.
3. Для добавления новой диаграммы достаточно в меню Diagram > New Diagram выбрать тип диаграммы Activity Diagram и нажмите кнопку Next. Задать имя диаграммы SaleSystemActivityDiagram и нажать ОК.
4. В результате отобразится рабочая область с элементами для построения диаграммы деятельности (рис. 11).

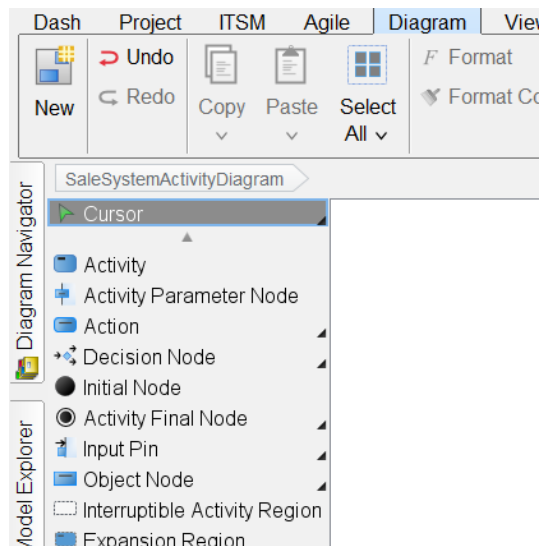


Рисунок 122 – Рабочая область для построения диаграммы деятельности

Далее будут описаны принципы построения диаграммы деятельности.

Для создания дорожек необходимо на панели инструментов выбрать **Vertical SwimLine** и затем курсором мыши выбрать место расположения на диаграмме (Рисунок 12). Чтобы добавить дорожку необходимо щелкнуть правой кнопкой мыши по элементу и в контекстном меню выбрать **Add Vertical Partition**. После создания дорожек необходимо задать их имена. Для это дважды щелкните по заголовку дорожки и введите имя (Рисунок 13).

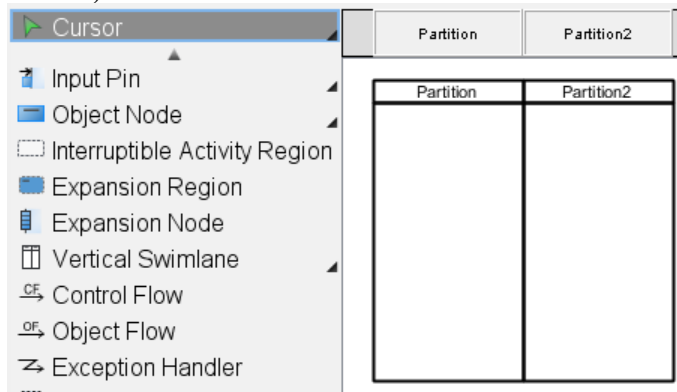


Рисунок 123 – Создание дорожек

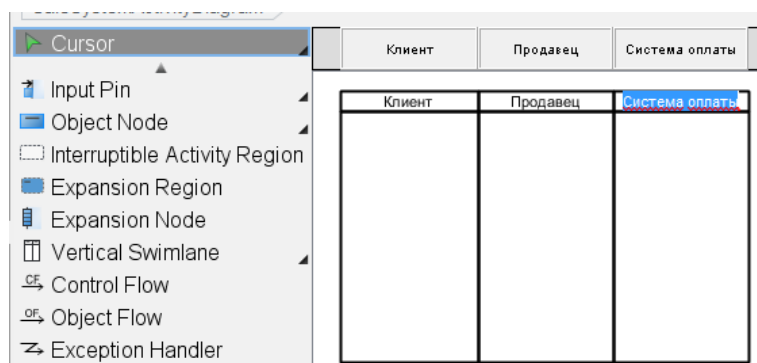


Рисунок 124 – Задание имени в созданных дорожках

Для добавления начального состояния на панели инструментов выберите **Initial Node** и поместите в нужное место рабочей области.

Для добавления нового состояния на панели инструментов выберите **Action** и поместите в нужное место рабочей области.

Для создания перехода между состояниями выберите **Control Flow > Action** (Рисунок 14).

Перетащите курсор мыши на пустое место на диаграмме для создания нового или на уже существующее состояние для соединения. Отпустите кнопку мыши (Рисунок 15).

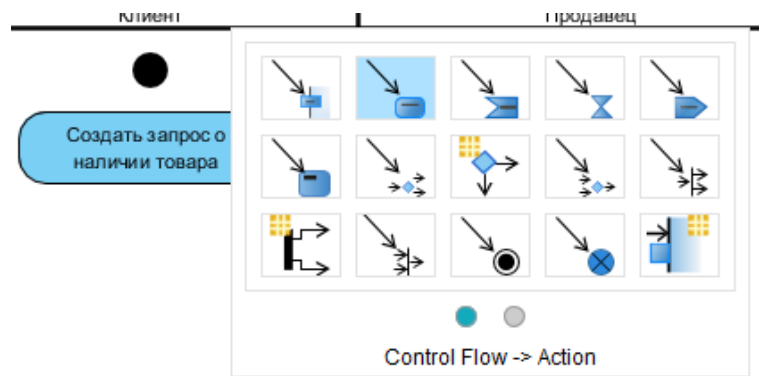


Рисунок 125 – Выбор типа перехода



Рисунок 126 – Создание перехода

Для добавления описания перехода дважды щелкните по нему и введите описание (рисунок 16).



Рисунок 127 – Добавление описания перехода

Чтобы добавить переход разветвления или объединения выберите **Control Flow > Fork Node** или **Join Node** (рисунок 17). После добавления элемента, для того чтобы изменить ориентацию на горизонтальную, щелкните правой кнопкой мыши по элементу и в контекстном меню выберите **Orientation > Horizontal**.

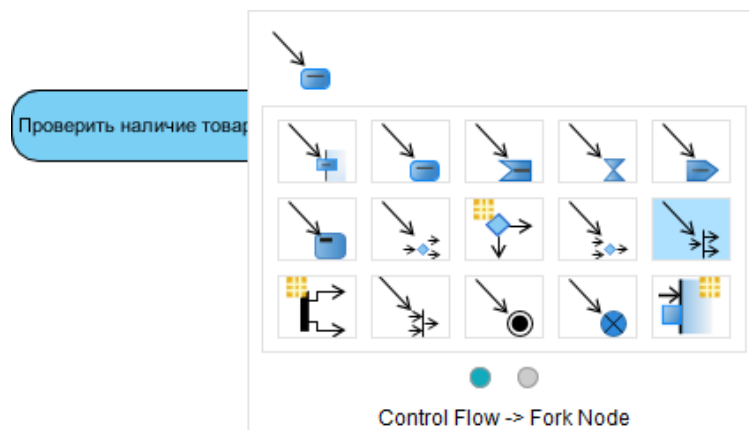


Рисунок 128 – Добавление перехода разветвления

Основываясь, на описанных выше принципах построения, постройте диаграммы деятельности для системы продажи товаров по каталогу в соответствии с рисунком 18.

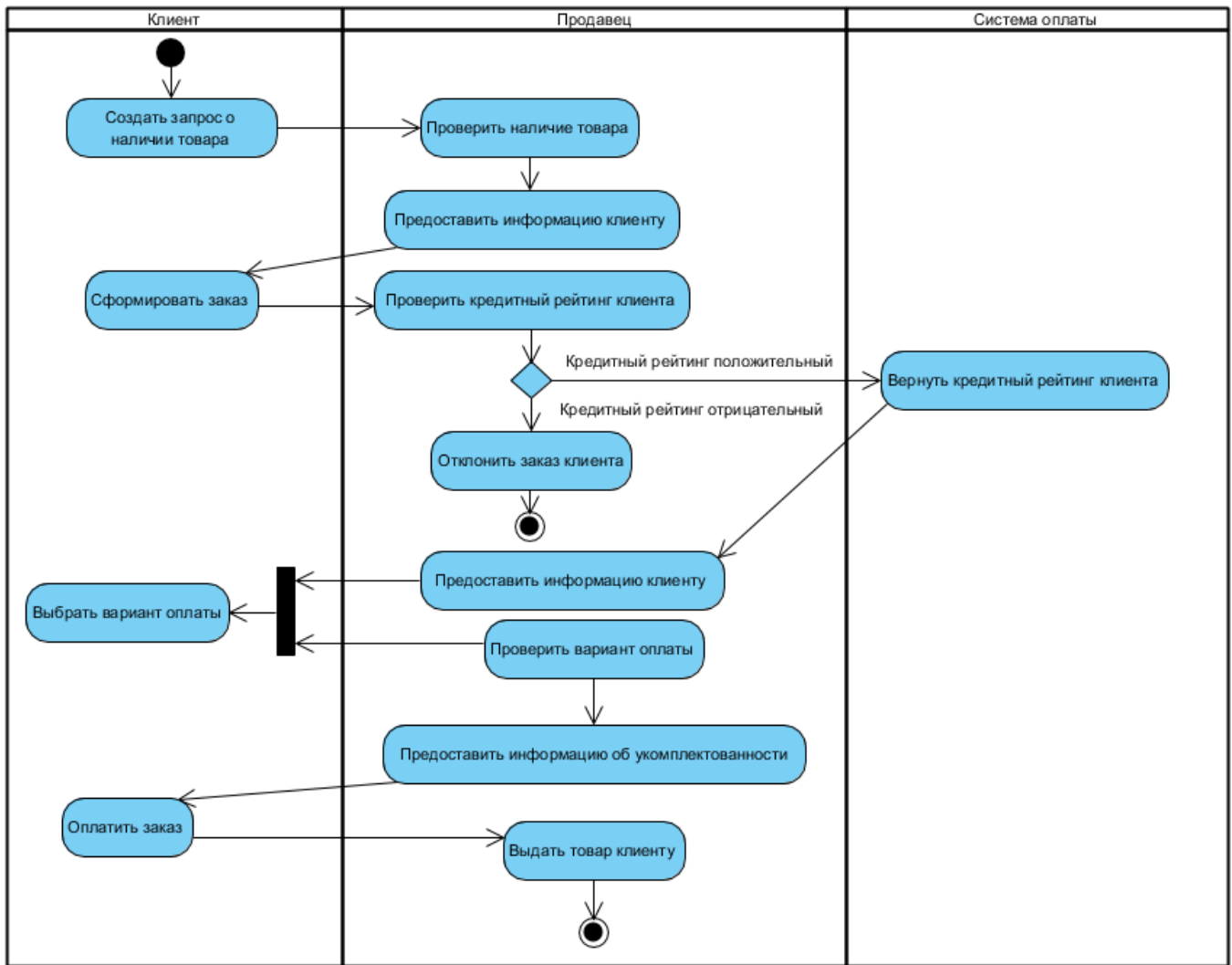


Рисунок 129 – Диаграмма деятельности системы продажи товаров по каталогу

4 Задание

Построить диаграмму деятельности в соответствии с вариантом.

Отчет по практическому занятию выполняется в формате MS Word, который содержит пошаговое описание процесса построения диаграммы, а также скриншоты результатов согласно заданию.

5 Варианты

1. «Отдел кадров»;
2. «Агентство аренды»;
3. «Аптека»;
4. «Ателье»;
5. «Аэропорт»;
6. «Библиотека»;
7. «Кинотеатр»;
8. «Поликлиника»;
9. «Автосалон»;
10. «Таксопарк».
11. «Издательство»;
12. «Прокат велосипедов»;
13. «Спортивный клуб».

6 Контрольные вопросы

1. Дайте определение понятию «диаграмма деятельности».
2. Опишите назначение диаграммы деятельности.

3. Дайте определение понятиям «состояние деятельности» и «состояние действия». Графическое изображение состояния.
4. Приведите пример ветвления и параллельных потоков управления процессами на диаграмме деятельности.
5. Какие переходы используются на диаграмме деятельности?
6. Что представляет собой дорожка на диаграмме деятельности?
7. Как графически изображаются объекты на диаграмме деятельности?

Практическая работа №22

Разработка диаграмм последовательности с помощью Visual Paradigm for UML

Цель: изучение основ создания диаграмм последовательности на языке UML с помощью case-средства Visual Paradigm for UML CE, применение приобретенных навыков для построения объектно-ориентированных моделей определенной предметной области.

1 Задачи

Основными задачами практической работы являются:

– получить навыки создания диаграмм последовательности с помощью case-средства Visual Paradigm for UML CE.

2 Краткие теоретические сведения

Диаграмма последовательности (sequence diagram) – диаграмма, на которой показаны взаимодействия объектов, упорядоченные по времени их проявления.

На диаграмме последовательности неявно присутствует ось времени, что позволяет визуализировать временные отношения между передаваемыми сообщениями. С помощью диаграммы последовательности можно представить взаимодействие элементов модели как своеобразный временной график «жизни» всей совокупности объектов, связанных между собой для реализации варианта использования программной системы, достижения цели или выполнения задачи.

На диаграмме последовательности изображаются объекты, которые непосредственно участвуют во взаимодействии, при этом никакие статические связи с другими объектами не визуализируются. Диаграмма последовательности имеет как бы два измерения: одно – слева направо в виде вертикальных линий, каждая из которых изображает линию жизни отдельного объекта; второе – вертикальная временная ось, направленная сверху вниз.

Каждый объект графически изображается в форме прямоугольника и располагается в верхней части своей линии жизни (рис. 1). Внутри прямоугольника записываются собственное имя объекта и имя класса, разделенные двоеточием, запись подчеркивается, что является признаком объекта, который, как указывалось ранее, представляет собой экземпляр класса.



Рисунок 130 – Графические элементы диаграммы последовательности

Объекты записываются так же, как и в диаграмме кооперации. **Анонимный объект** – такой объект, для которого отсутствует собственное имя, но указано имя класса. **Объект-сирота** – такой объект, для которого может отсутствовать имя класса, но указано собственное имя объекта. Роль классов в именах объектов на диаграммах последовательности, как правило, не указывается.

Крайним слева на диаграмме изображается объект-инициатор моделируемого процесса взаимодействия (объект а на рис. 1). Правее – другой объект, который непосредственно взаимодействует с первым, т.е. порядок расположения объектов на диаграмме последовательности определяется исключительно соображениями удобства визуализации их взаимодействия друг с другом.

Начальному моменту времени соответствует самая верхняя часть диаграммы. При этом процесс взаимодействия объектов реализуется посредством сообщений, которые посылаются одними объектами другим. Сообщения изображаются в виде горизонтальных стрелок с именем сообщения и образуют определенный порядок относительно времени своей инициализации. Другими словами, сообщения,

расположенные на диаграмме последовательности выше, передаются раньше тех, которые расположены ниже. При этом масштаб на оси времени не указывается, поскольку диаграмма последовательности моделирует лишь временную упорядоченность взаимодействий типа «раньше-позже».

Линия жизни объекта (object lifeline) – вертикальная линия на диаграмме последовательности, которая представляет существование объекта в течение определенного периода времени. Линия жизни объекта изображается пунктирной вертикальной линией, ассоциированной с единственным объектом на диаграмме последовательности. Линия жизни служит для обозначения периода времени, в течение которого объект существует в системе и может потенциально участвовать во всех ее взаимодействиях. Если объект существует в системе постоянно, то и его линия жизни должна продолжаться по всей рабочей области диаграммы последовательности от самой верхней ее части до самой нижней («объект 1» и анонимный объект «Класса 2»).

Отдельные объекты, закончив выполнение своих операций, могут быть уничтожены, чтобы освободить занимаемые ими ресурсы. Для таких объектов линия жизни обрывается в момент его уничтожения. Для обозначения момента уничтожения объекта в языке UML применяется специальный символ в форме латинской буквы «X». На рис. 2 этот символ используется для уничтожения анонимного объекта, образованного от «Класса 3». Ниже этого символа пунктирная линия не изображается, поскольку соответствующего объекта в системе уже нет, и этот объект должен быть исключен из всех последующих взаимодействий.

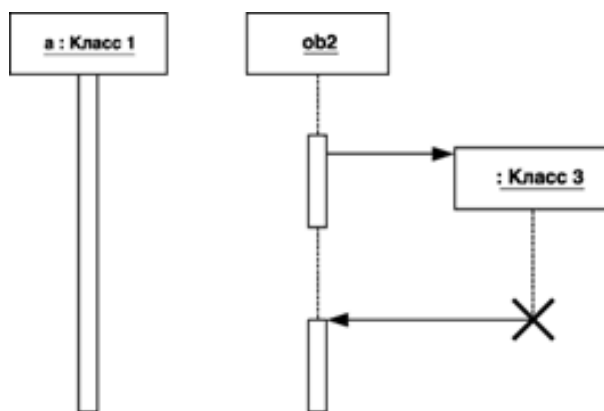


Рисунок 131 – Графическое изображение линий жизни и фокусов управления объектов

Отдельные объекты в системе могут создаваться по мере необходимости, существенно экономя ресурсы системы и повышая ее производительность. В этом случае прямоугольник такого объекта изображается не в верхней части диаграммы последовательности, а в той, которая соответствует моменту создания объекта (анонимный объект, образованный от «Класса 3» на рис. 2). При этом прямоугольник объекта вертикально располагается в том месте диаграммы, которое по оси времени совпадает с моментом его возникновения в системе. Объект создается со своей линией жизни, а, возможно, и с фокусом управления.

В процессе функционирования объектно-ориентированных систем одни объекты могут находиться в активном состоянии, непосредственно выполняя определенные действия, или в состоянии пассивного ожидания сообщений от других объектов. Фокус управления – символ, применяемый для того, чтобы явно выделить подобную активность объектов на диаграммах последовательности.

Фокус управления (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 в форме вытянутого узкого прямоугольника («объект а» на рис. 1), верхняя сторона которого обозначает начало получения фокуса управления объектом (начало активности), а ее нижняя сторона – окончание фокуса управления (окончание активности). Этот прямоугольник располагается ниже обозначения соответствующего объекта и может заменять его линию жизни («объект а» на рис. 2), если на всем ее протяжении он активен.

Периоды активности объекта могут чередоваться с периодами его пассивности или ожидания. В этом случае у такого объекта фокусы управления изменяют свое изображение на линию жизни и наоборот (объект сирота «ob2» на рис. 2). Получить фокус управления может только объект, у которого в этот момент имеется линия жизни. Если же объект был уничтожен, то вновь возникнуть в системе он уже не может. Вместо него может быть создан лишь экземпляр этого же класса, который, строго говоря, будет другим объектом.

В отдельных случаях объект может посылать сообщения самому себе, инициируя так называемые *рефлексивные* сообщения. Для этой цели служит специальное изображение (сообщение у «объекта а» на рис. 3). Такие сообщения изображаются в форме сообщения, начало и конец которого соприкасаются с линией жизни или фокусом управления одного и того же объекта. Подобные ситуации возникают, например, при обработке нажатий на клавиши клавиатуры при вводе текста в редактируемый документ, при наборе цифр номера телефона абонента.

Если в результате рефлексивного сообщения создается новый подпроцесс или нить управления, то говорят о рекурсивном или вложенном фокусе управления. На диаграмме последовательности рекурсия обозначается небольшим прямоугольником, присоединенным к правой стороне фокуса управления того объекта, для которого изображается данное рекурсивное взаимодействие (анонимный объект «Класса 2» на рис. 3).



Рисунок 132 – Графическое изображение актера, рефлексивного сообщения и рекурсии

Сообщения на диаграмме последовательности имеют дополнительные семантические особенности. На диаграмме последовательности все сообщения упорядочены по времени своей передачи в моделируемой системе.

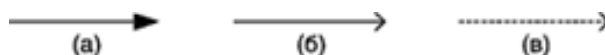


Рисунок 133 – Графическое изображение различных видов сообщений между объектами на диаграмме последовательности

Первая разновидность сообщения (рис. 4, а) наиболее распространена и используется для **вызова процедур, выполнения операций или обозначения отдельных вложенных потоков управления**. Начало этой стрелки, как правило, соприкасается с фокусом управления того объекта-клиента, который инициирует это сообщение. Конец стрелки соприкасается с линией жизни того объекта, который принимает это сообщение и выполняет в ответ определенные действия. При этом принимающий объект может получить фокус управления, становясь в этом случае активным. Передающий объект может потерять фокус управления или остаться активным.

Вторая разновидность сообщения (рис. 4, б) используется для обозначения **простого асинхронного сообщения**, которое передается в произвольный момент времени. Передача такого сообщения обычно не сопровождается получением фокуса управления объектом-получателем.

Третья разновидность сообщения (рис. 4, в) используется для **возврата из вызова процедуры**. Примером может служить простое сообщение о завершении вычислений без предоставления результата расчетов объекту-клиенту. В процедурных потоках управления эта стрелка может быть опущена, поскольку ее наличие неявно предполагается в конце активизации объекта. В то же время считается, что каждый вызов процедуры имеет свою пару – возврат вызова. Для непроцедурных потоков управления, включая параллельные и асинхронные сообщения, стрелка возврата должна указываться явным образом.

Каждое сообщение на диаграмме последовательности ассоциируется с определенной операцией, которая должна быть выполнена принявшим его объектом. При этом операция может иметь аргументы или параметры, значения которых влияют на получение различных результатов. Соответствующие параметры операции будет иметь и вызывающее это действие сообщение. Более того, значения параметров отдельных сообщений могут содержать условные выражения, образуя ветвление или альтернативные пути основного потока управления.

Одна из **особенностей диаграммы последовательности** – возможность визуализировать простое *ветвление процесса*. Для изображения ветвления используются две или более стрелки, выходящие из

одной точки фокуса управления объекта (объект «ob1» на рис. 5). При этом рядом с каждой из них должно быть явно указано соответствующее условие ветви в форме булевого выражения.

Количество ветвей может быть произвольным, но наличие ветвлений может существенно усложнить интерпретацию диаграммы последовательности. Предложение-условие должно быть явно указано для каждой ветви и записывается в форме обычного текста, псевдокода или выражения языка программирования. Это выражение всегда должно возвращать некоторое булево выражение. Запись этих условий должна исключать одновременную передачу альтернативных сообщений по двум и более ветвям. В противном случае на диаграмме последовательности может возникнуть конфликт ветвления.

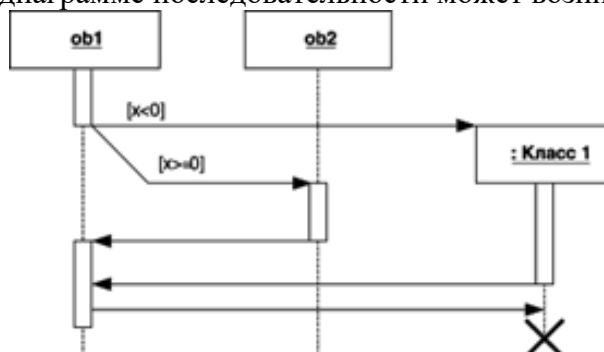


Рисунок 134 – Графическое изображение бинарного ветвления потока управления на диаграмме последовательности

Пример: объект «ob1» вызывает выполнение действий у одного из трех других объектов. Условием ветвления может служить сумма снимаемых клиентом средств со своего текущего счета. Если эта сумма превышает 1500\$, то могут потребоваться дополнительные действия, связанные с созданием и последующим разрушением объекта «Класса 1». Если же сумма превышает 100\$, но не превышает 1500\$, то вызывается операция или процедура объекта «ob3». И, наконец, если сумма не превышает 100\$, то вызывается операция или процедура объекта «ob2». При этом объекты «ob1», «ob2» и «ob3» постоянно существуют в системе. Последний объект создается от «Класса 1» только в том случае, если справедливо первое из альтернативных условий. В противном случае он может быть никогда не создан.

Объект «ob1» имеет постоянный фокус управления, а все остальные объекты – получают фокус управления только для выполнения ими соответствующих операций. На диаграммах последовательности при записи сообщений также могут использоваться стереотипы. Ниже представлена диаграмма последовательности для описанного выше случая ветвления, дополненная стереотипными значениями отдельных сообщений (рис. 6).

Сообщения могут иметь собственное имя, в качестве которого выступает имя операции, вызов которой инициируют эти сообщения у принимающего объекта. Тогда со стрелкой записывается имя операции с круглыми скобками, в которых могут указываться параметры или аргументы соответствующей операции. Если параметры отсутствуют, то скобки все равно изображаются. Построение диаграммы последовательности начинают с выделения из всей совокупности классов только тех, объекты которых участвуют во взаимодействии. После все объекты наносятся на диаграмму, с соблюдением порядка инициализации сообщений.

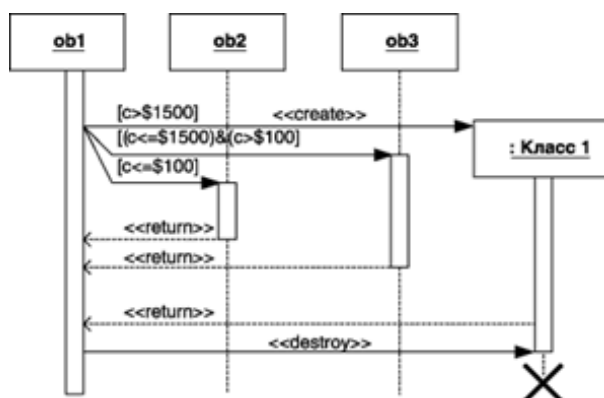


Рисунок 135 – Диаграмма последовательности со стереотипными значениями сообщений

Когда объекты визуализированы, можно приступать к спецификации сообщений, учитывая те операции, которые имеют классы соответствующих объектов, иногда используя их стереотипы.

Для уничтожения объектов, которые создаются на время выполнения своих действий, нужно предусмотреть явное сообщение. Наиболее простые случаи ветвления процесса взаимодействия можно изобразить на одной диаграмме, а в более сложных случаях для моделирования каждой ветви управления может потребоваться отдельная диаграмма последовательности. Общим правилом является визуализация особенностей реализации каждого варианта использования на отдельной диаграмме последовательности.

Пример: Программное средство представляет среду для формирования отчетов по лекционному материалу. Пользователь может вводить свою информацию в отчеты, изменять параметры. После оформления отчетов пользователю предоставлена возможность сохранения отчета. Вывода его на печать и его архивацию.

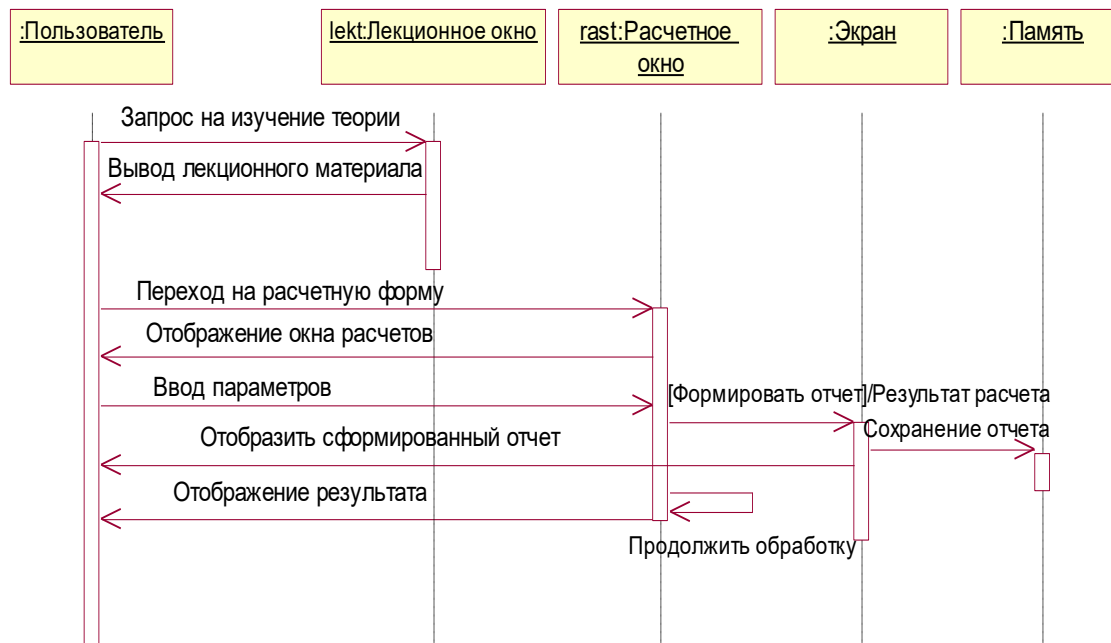


Рисунок 136 – Пример диаграммы последовательности

3 Методика выполнения

В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

1. Запустите Visual Paradigm for UML CE.
2. Откройте проект, содержащий диаграмму вариантов использования, построенную в практическом занятии №15.
3. Для добавления новой диаграммы достаточно в меню Diagram > New Diagram выбрать тип диаграммы Sequence Diagram и нажмите кнопку Next. Задать имя диаграммы SaleSystem SequenceDiagram и нажать ОК.
4. В результате отобразится пустая рабочая область с элементами для построения диаграммы последовательности (рис. 8).

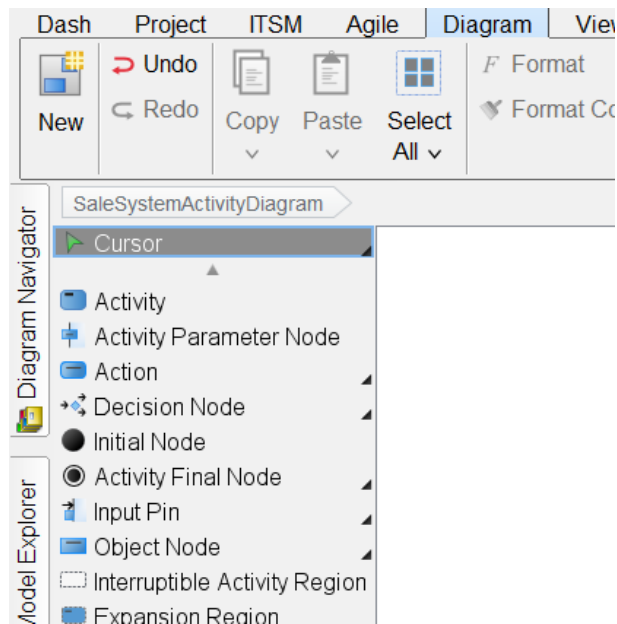


Рисунок 137 – Рабочая область для построения диаграммы последовательности

Далее будут описаны принципы построения диаграммы последовательности.

Для того чтобы создать актера, щелкните кнопкой мыши **Actor** на панели инструментов и затем щелкните на диаграмме (Рисунок 9).



Рисунок 138 – Создание Actor

Для создания линии жизни объекта нужно щелкните кнопкой мыши **LifeLine** на панели инструментов и затем щелкните на диаграмме. Другим более быстрым способом создания линии жизни объекта является способ создание через интерфейс ресурса (Рисунок 10). Нужно выбрать **Message > LifeLine** и перетащить курсор на пустое место на диаграмме, далее отпустить кнопку (Рисунок 11).

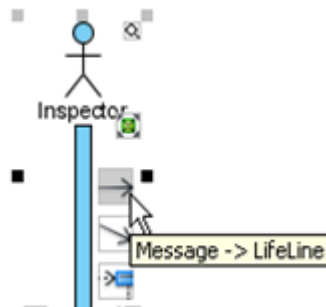


Рисунок 139 – Создание линии жизни объекта



Рисунок 140 – Результат создания линии жизни объекта

Для того чтобы сделать диаграмму более читабельной можно воспользоваться инструментом **Sweeper**, позволяющий раздвигать элементы диаграммы последовательности (Рисунок 12).

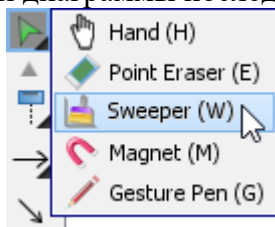


Рисунок 141 – Инструмент Sweeper

После выбора инструмента **Sweeper** перетащите его в нужном направлении, в этом же направлении будут перемещены и элементы диаграммы, находящиеся по другую сторону от **Sweeper** (Рисунок 13).

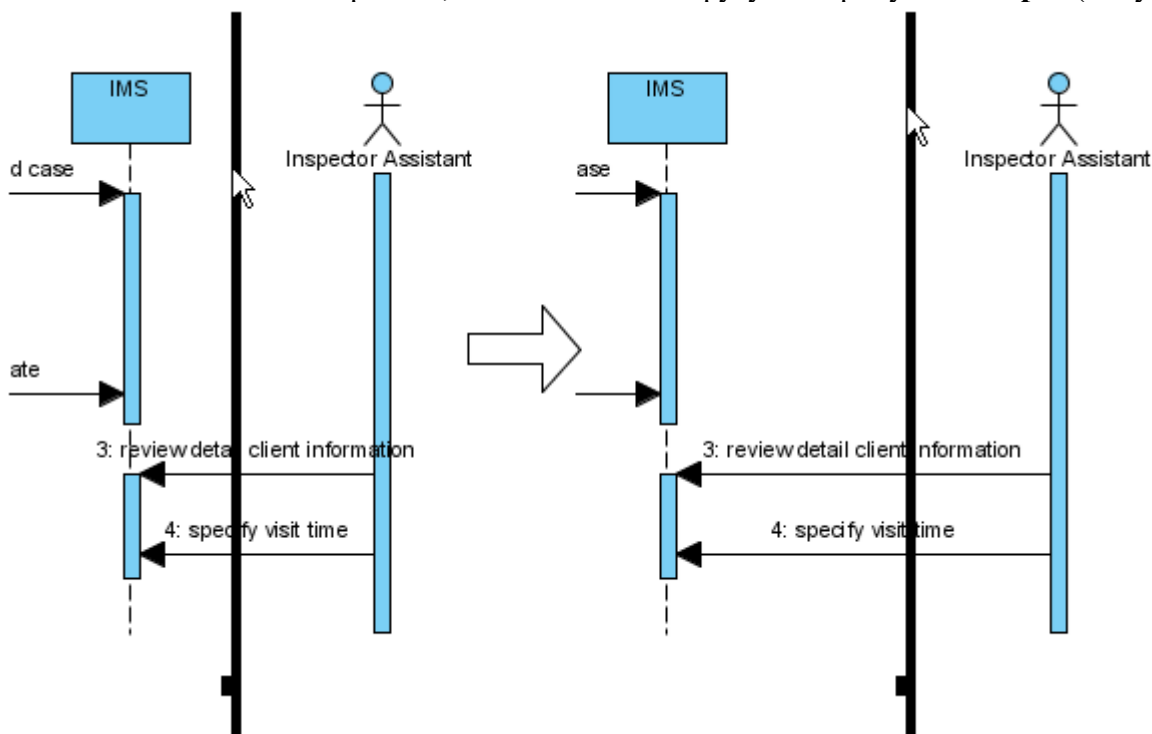


Рисунок 142 – Смещение элементов диаграммы последовательности вправо

Существует два способа нумерации сообщений **Diagram-based** и **Frame-based**.

При нумерации способом **Diagram-based** необходимо щелкнуть правой клавишей мыши на листе диаграммы и выбрать **Sequence Number**, а затем **Single Level** или **Nested Level** (Рисунок 14).

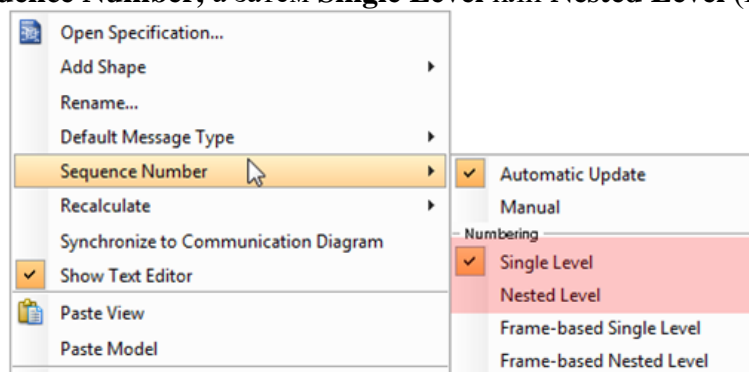


Рисунок 143 – Всплывающее меню при выборе нумерации сообщений Diagram-based способом

Если был выбран **Single Level**, то нумерация будет производиться целыми числами, при выборе **Nested Level** способа – числами с десятичными знаками (Рисунок 15).

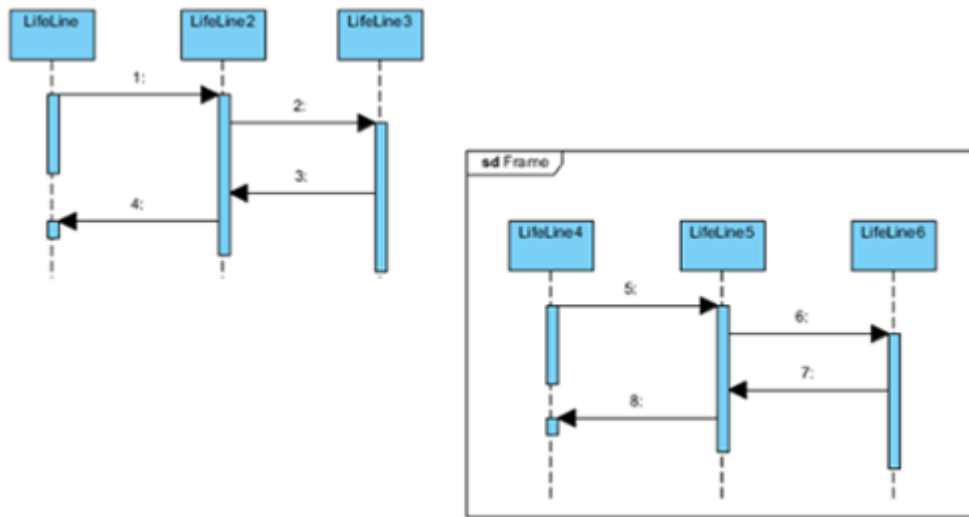


Рисунок 144 – Нумерация сообщений вариантом Diagram-based Single Level

При нумерации способом **Frame-based** необходимо щелкнуть правой клавишей мыши на листе диаграммы и выбрать **Sequence Number**, а затем **Frame-based Single Level** или **Frame-based Nested Level** (Рисунок 16).

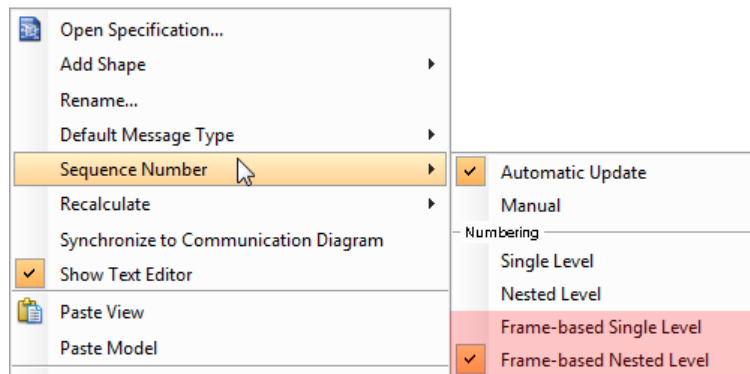


Рисунок 145 – Всплывающее меню при выборе нумерации сообщений Frame-based способом

При выборе нумерации сообщений способом **Frame-based** нумерация каждый раз начинается сначала в каждой независимой области (Рисунок 17).

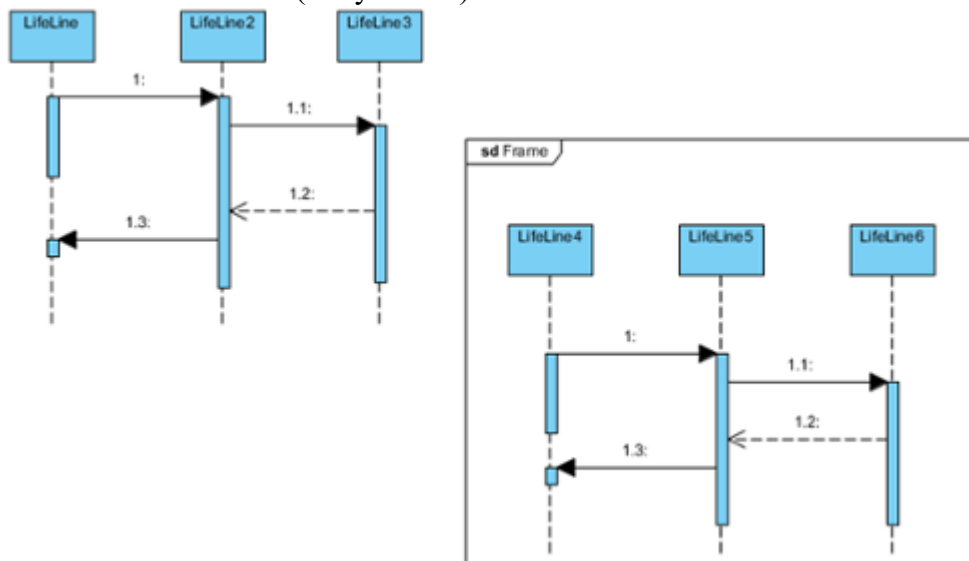


Рисунок 146 – Нумерация сообщений вариантом Frame-based Nested Level

Основываясь, на описанных выше принципах построения, постройте диаграммы последовательности для всех вариантов использования системы продажи товаров по каталогу в соответствии с рисунками 18-

Для добавления диаграммы последовательности к варианту использования откройте диаграмму прецедентов, щелкните правой кнопкой мыши по варианту использования и в контекстном меню

выберите **Sub Diagrams > New Diagram ...**. Затем в окне выбора диаграммы найдите **Sequence Diagram** и нажмите ОК.

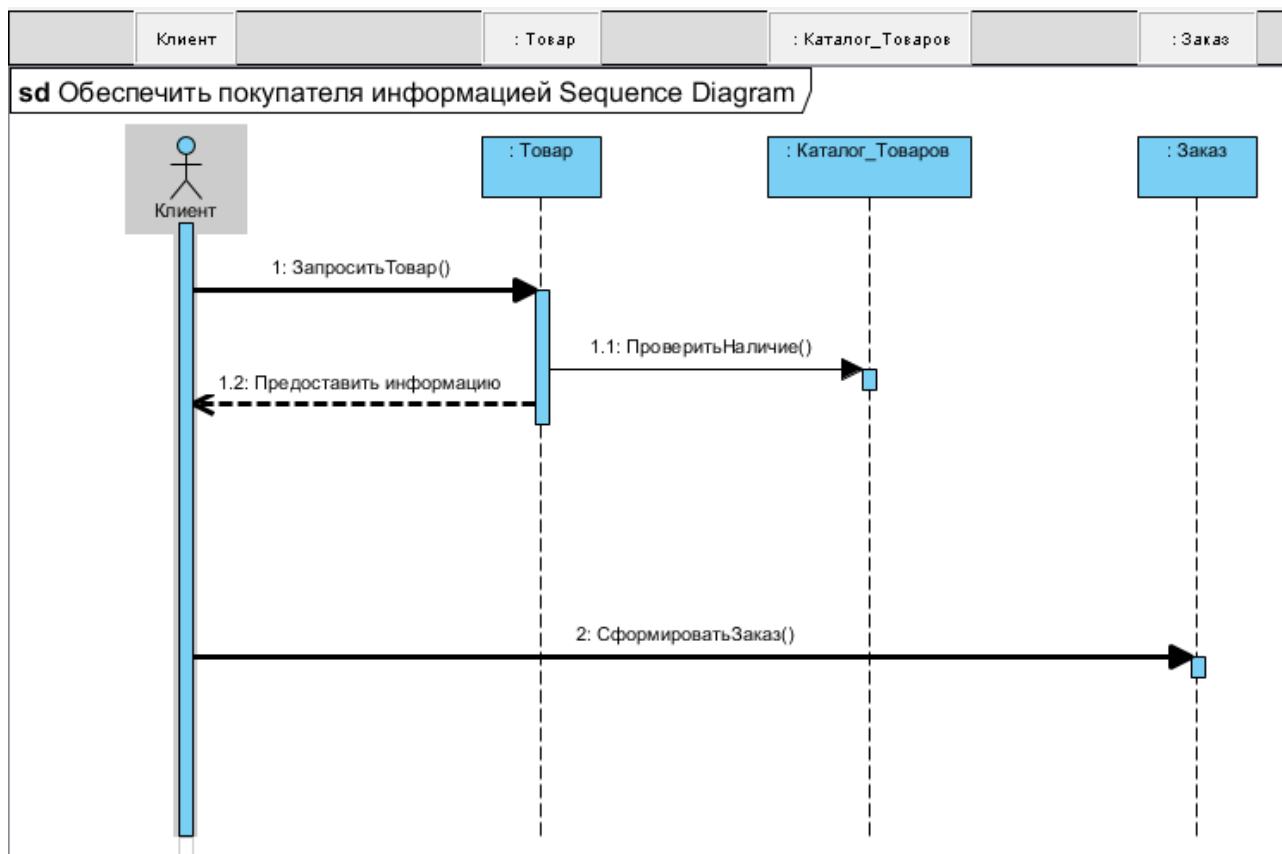


Рисунок 147 – Диаграмма последовательности для варианта использования «Обеспечить покупателя информацией»

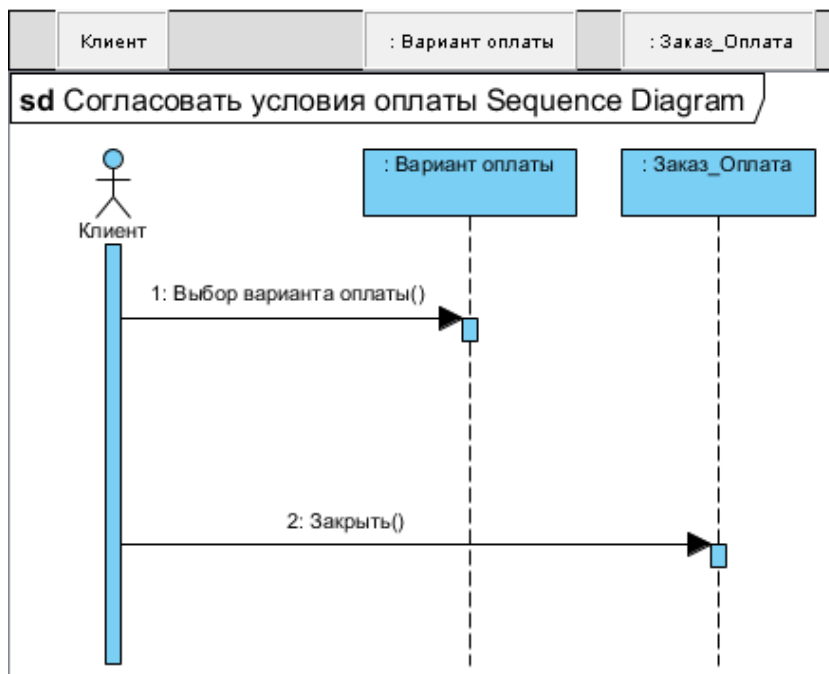


Рисунок 148 – Диаграмма последовательности для варианта использования «Согласовать условия оплаты»

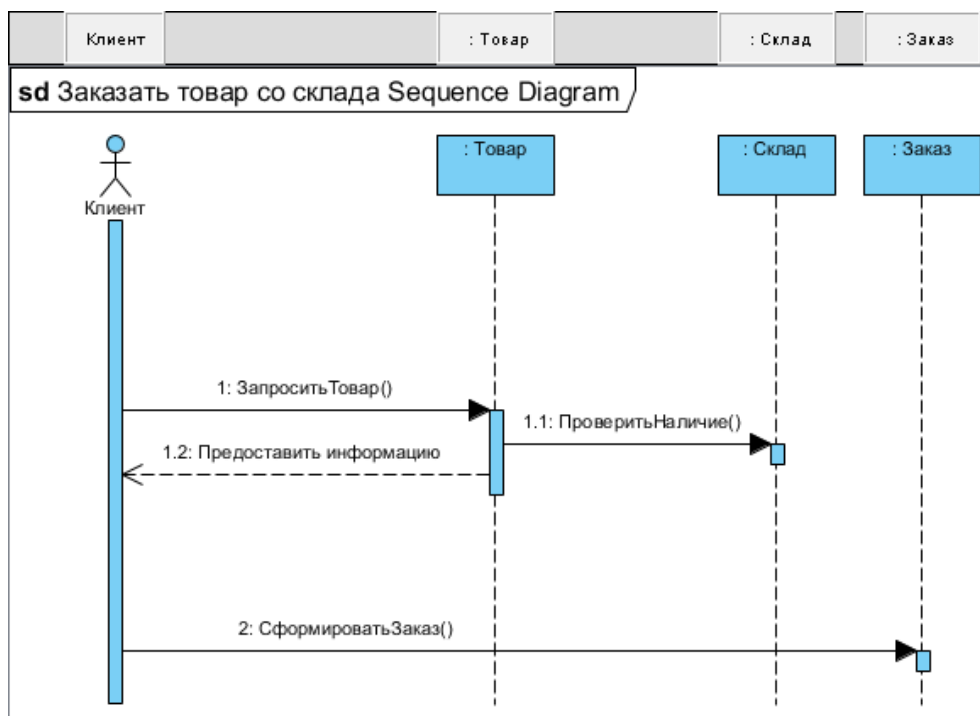


Рисунок 149 – Диаграмма последовательности для прецедента «Заказать товар со склада»

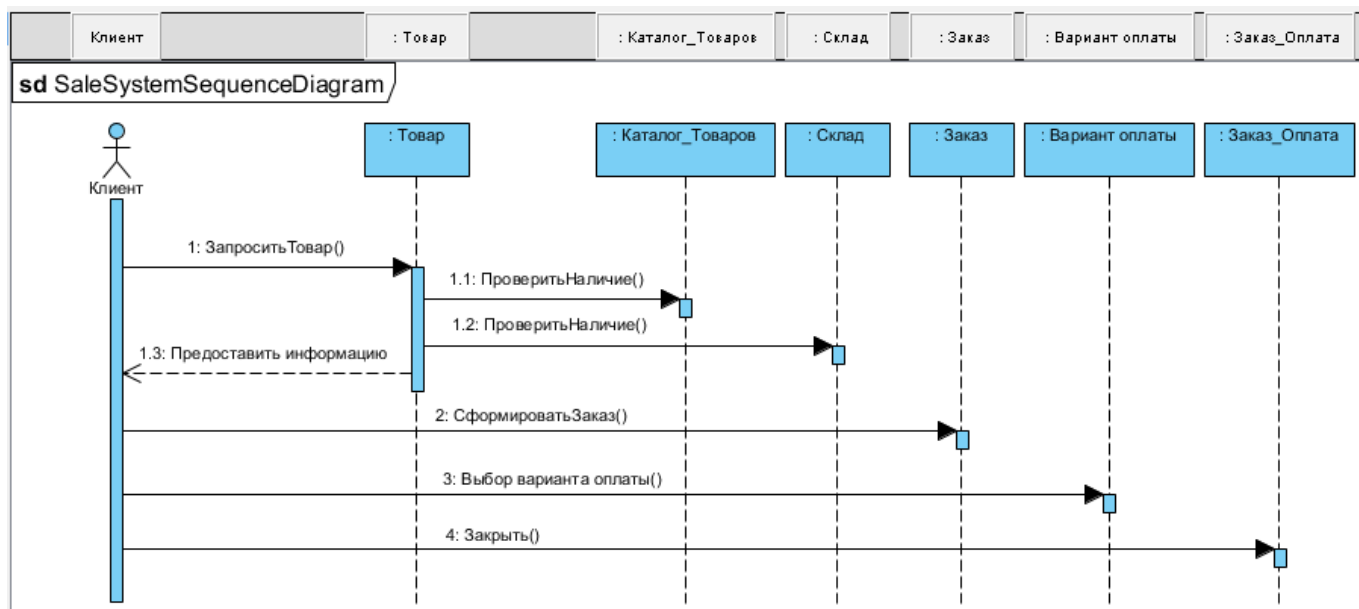


Рисунок 150 – Диаграмма последовательности для системы продажи товаров по каталогу

4 Задание

Построить диаграммы последовательности для всех вариантов использования (Практическая работа №18) и для всей системы в целом в соответствии с вариантом.

Отчет по практическому занятию выполняется в формате MS Word, который содержит пошаговое описание процесса построения диаграммы, а также скриншоты результатов согласно заданию.

5 Варианты

1. «Отдел кадров»;
2. «Агентство аренды»;
3. «Аптека»;
4. «Ателье»;
5. «Аэропорт»;
6. «Библиотека»;
7. «Кинотеатр»;

8. «Поликлиника»;
9. «Автосалон»;
10. «Таксопарк».
11. «Издательство»;
12. «Прокат велосипедов»;
13. «Спортивный клуб».

6 Контрольные вопросы

1. Дайте определение понятию «диаграмма последовательности».
2. Опишите назначение диаграммы последовательности.
3. В чем различие анонимного объекта и объекта-сироты?
4. Охарактеризуйте линию жизни объекта. Ее графическое изображение.
5. Для чего предназначен фокус управления? Его графическое изображение.
6. Дайте определение понятию «сообщение». Какие виды сообщений используются в диаграмме последовательности? Когда используется рефлексивное сообщение?
7. Расскажите про ветвления на диаграмме последовательности. Приведите пример ветвления.