Компания «Информационно-вычислительные Системы»

Технологический отдел

Лабораторные работы по CASEBERRY

№ 1, 2

Раздел: Обучение

ТЕХНОЛОГИЧЕСКИЙ ОТДЕЛ

Лабораторные работы по CASEBERRY

Компания ИВС
г Пермь 614007 ул. Островского 65
Тел.: (3422) 385-207, 196-500 Факс: (3422) 196-510

Лабораторная работа №1

Построение UML-диаграмм

Цель работы: получить навыки построения UML-диаграмм в среде CASEBERRY.

Использование объектно-ориентированного подхода при проектировании систем

Большинство современных методов объектно-ориентированного проектирования основаны на использовании языка UML.

Унифицированный язык моделирования (UML, Unified Model Language) является приемником языков и методов объектно-ориентированного анализа и проектирования, которые появились в конце 80-х и начале 90-х. Он непосредственно унифицирует методы Буча, Рембо и Джекобсона, однако обладает большими возможностями. Язык моделирования UML прошел процесс стандартизации в рамках консорциума OMG (Object Management Group) и в настоящее время является стандартом OMG.

UML представляет собой универсальный язык для анализа предметных областей, существующих систем, моделирования систем документирования объектных моделей, проектирования программного обеспечения. На UML можно содержательно описывать классы, объекты и компоненты в различных предметных областях, часто сильно отличающихся друг от друга.

В процессе разработки система представляется в виде объединения нескольких проекций, каждая из которых описывает определенный аспект разрабатываемой системы, а вместе они определяют систему во всей ее полноте.

CASEBERRY

Краткое описание

Ключевой идеей технологии CASEBERRY, является поддержка всего жизненного цикла разработки программного обеспечения через единую, от анализа до кода, модель информационной системы, связанную на всех этапах. Непосредственно с моделей в нотации UML автоматизировано создаются заготовки исходных кодов, проработанные до вполне завершённых, включая и пользовательский интерфейс, приложений, которые компилируются и запускаются непосредственно из среды комплекса, что выгодно отличает его от большинства аналогичных CASE - средств.

Такой подход позволяет значительно сократить сроки производства ПО, освободить от рутины программирования разработчиков, сосредоточив их усилия на реализации сугубо прикладных задач, а также избежать многих архитектурных ошибок. Кроме того, можно в сжатые сроки разрабатывать прототипы для оперативного предъявления заказчику на начальных стадиях проекта.

Комплекс включает в себя:

- Инструмент объектно-ориентированного проектирования (средство создания диаграмм);
- Инструменты автоматизированного создания исходного кода систем и баз данных, а также библиотеки для программистов.
- Репозитарий моделей, который имеет чёткую структуру, не ограничивающую, вместе с тем, проектировщика какой-либо одной, предопределённой, методикой. Например, можно создать любое количество систем, конфигураций, стадий и поддерживать жизненный цикл

проекта. Репозитарий может быть расширен с целью добавления другой функциональности, а также для генерации исходного кода при помощи механизма надстроек (PlugIn).

Разработчикам ПО комплекс помогает решать и автоматизировать множество практических задач, таких, как:

- Создание на уровне кода классов и объектов, соответствующих предметным сущностям и их отношениям;
- ORM (Object-Relational Mapping) - объектно-реляционное отображение, хранение объектных данных в реляционных БД, в том числе объектов наследующихся классов;
- Поддержка различных реляционных СУБД и поддержка источников данных любой другой «природы»;
- Создание пользовательского интерфейса;
- Реализация системной архитектуры: от монолитной до распределённой многоуровневой;

Начало выполнения лабораторной работы

1. Запустите CASEBERRY
2. Выберите репозитарий и добавьте новый проект. Назовите его «АСУ Склад <Ваша ФИО>»

3. Создайте конфигурацию, стадию и систему

Постановка задачи

Характеристики объекта автоматизации

Предприятие производит оптовую реализацию промышленной продукции широкого ассортимента. Поставщиками компании выступают заводы и фабрики, находящиеся на территории РФ. Клиенты предприятия – предприниматели, фирмы и др. организации, осуществляющие розничную и мелкооптовую продажу.

Постановка задачи на проектирование информационной системы

Проектируемая система обязана производить учет и контроль движения продуктов на складе. Автоматизировать процесс выписки накладных, счётов и других документов.

Описание бизнес-процесса «Реализация продукции со склада»

Клиент, решивший оформить заказ на поставку продукции, обращается в офис предприятия. Менеджер согласовывает с клиентом все условия по оформлению заказа. При этом менеджер обязан проверить наличие на складе каждого из заявленных продуктов.

В случае если все затребованные клиентом позиции продуктов есть в наличии, либо клиентом приняты альтернативные варианты, заказ передается в бухгалтерию, и клиенту предлагается его оплатить.

Если клиент оплачивает заказ по наличному расчету, то после оплаты бухгалтер сразу выписывает две товарно-транспортные накладные, которые передаются клиенту.

После получения накладной клиент прибывает на склад за своим товаром. Кладовщик выдает необходимые продукты и делает отметку в обоих экземплярах накладной о том, что груз выдан. Далее, клиент расписывается в двух экземплярах накладной и отбывает с

полученным товаром и одним экземпляром накладной. Второй экземпляр накладной остается у кладовщика.

Построение диаграммы вариантов использования (Usecase Diagram)

Краткие сведения о диаграмме вариантов использования.

Диаграмма вариантов использования является самым общим представлением функциональных требований к системе. Для последующего проектирования системы требуются более конкретные детали, которые описываются в документе, называемом «сценарием варианта использования» или «потоком событий» (flow of events). Сценарий подробно документирует процесс взаимодействия действующего лица с системой, реализуемого в рамках варианта использования. Основной поток событий описывает нормальный ход событий (при отсутствии ошибок). Альтернативные потоки описывают отклонения от нормального хода событий (ошибочные ситуации) и их обработку.

Достоинства модели вариантов использования заключаются в том, что она:

- определяет пользователей и границы системы;
- определяет системный интерфейс;
- удобна для общения пользователей с разработчиками;
- используется для написания тестов;
- является основой для написания пользовательской документации;
- хорошо вписывается в любые методы проектирования (как объектно-ориентированные, так и структурные).

Основные элементы диаграмм вариантов использования

Активный субъект (actor) отождествляется с чем-то или с кем-то, взаимодействующим с системой, т.е. играет определённую роль по отношению к системе, это может быть не обязательно пользователь будущей системы, так же это может быть внешняя система.

Варианты использования (use cases) позволяют моделировать диалог между активным субъектом и системой и отображают функции системы. С каждым вариантом использования связан определенный поток событий, происходящих по мере выполнения соответствующих функций системы. При описании потока событий определяется, что необходимо осуществить, и игнорируются аспекты того, как это делается.

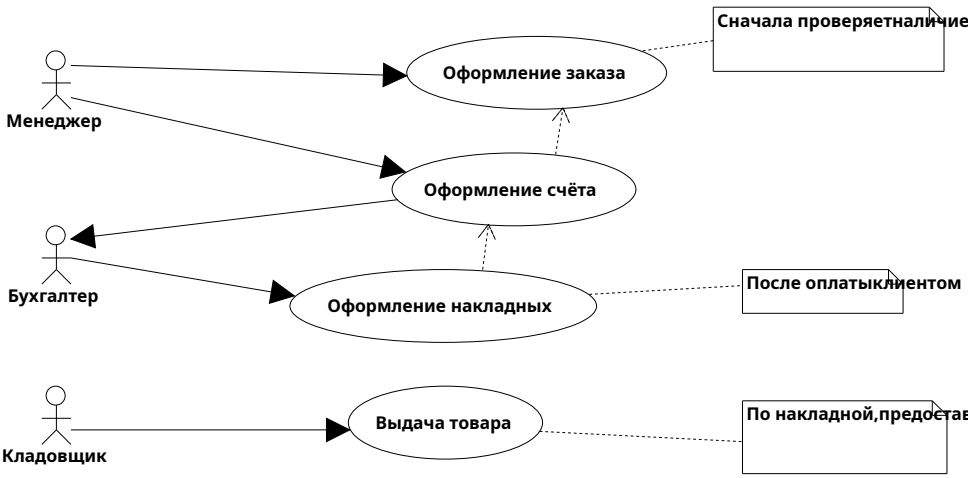
Между активным субъектом и вариантом использования устанавливаются связь *ассоциация (association relationship),* которая выполняет коммуникативную функцию, сообщая о взаимодействии субъекта с системой в рамках определенного варианта использования. Направление связи указывает, кто (субъект или система) является инициатором взаимодействия.

Помимо связей между субъектом и вариантом использования, связи могут устанавливаться и между вариантами использования. Связи бывают двух типов - *включающими (inclusive) и расширяющими (extensive).*

Порядок построения Usecase Diagram

1. Создайте usecase диаграмму с именем «Основная функциональность»
2. Проанализируйте какие активные субъекты должны взаимодействовать с будущей системой.

3. Нарисуйте actor'ов. (Предлагается сделать 3: Менеджер, Бухгалтер и Кладовщик.)
4. Добавьте следующие прецеденты:
   a. Оформление заказа
   b. Оформление счёта
   c. Оформление накладной
   d. Выдача товара
5. Для пояснения можно использовать комментарии
6. Расставьте стрелки, обозначающие зависимость (подумайте какие прецеденты находятся в отношении зависимости)
7. Должна быть получена подобная диаграмма:



8. Сохраните диаграмму

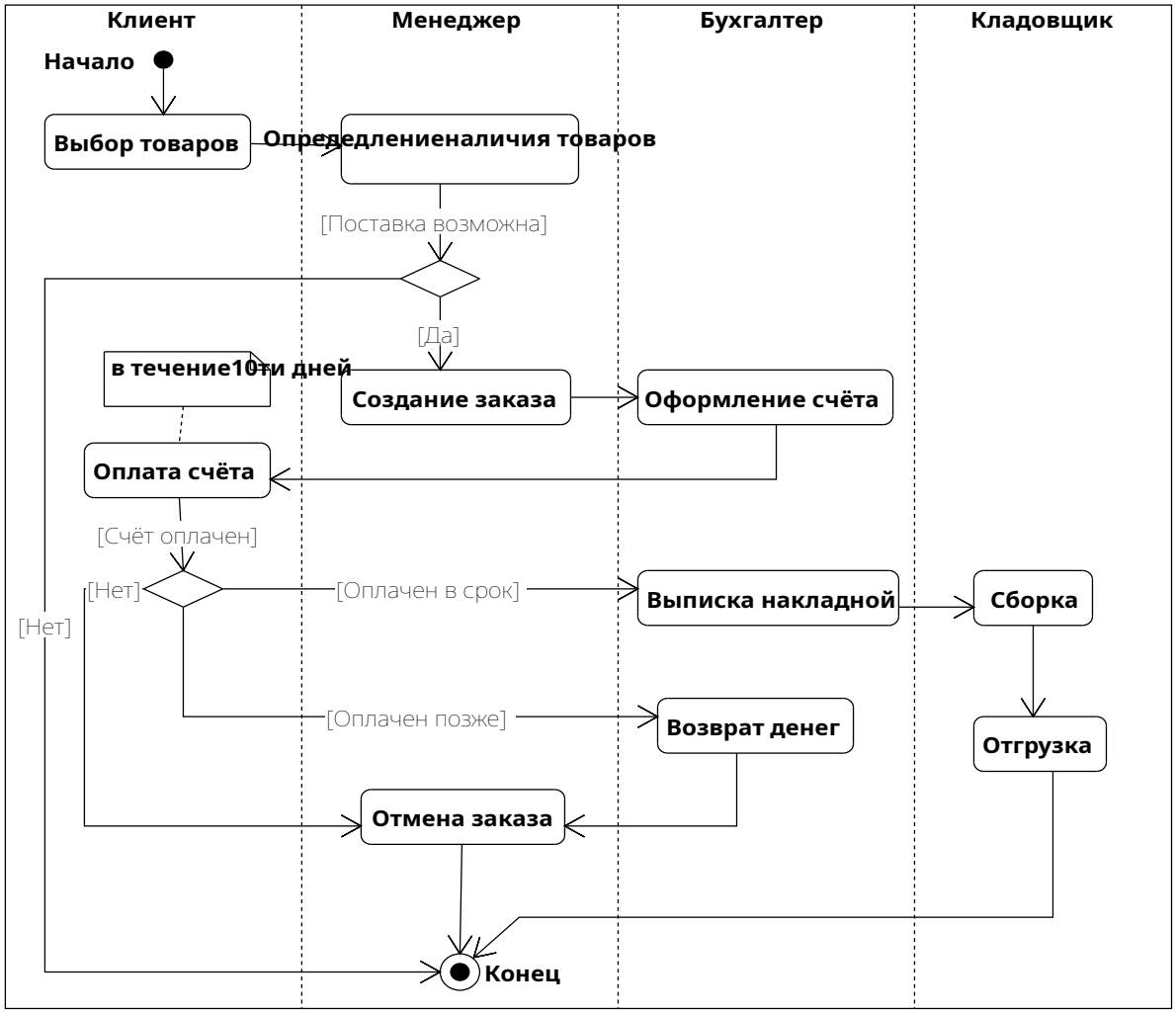
Построение диаграммы видов деятельности (Activity Diagram)

Краткие теоретические сведения

Диаграмма видов деятельности, как и диаграмма состояний, отражает динамические аспекты поведения системы. По существу, эта диаграмма представляет собой блок-схему, которая наглядно показывает, как поток управления переходит от одной деятельности к другой. Рассмотрим укрупненную диаграмму деятельности для описания процесса реализации продукции клиенту. Это позволит лучше разобраться в происходящих действиях.

Порядок построения диаграммы видов деятельности

1. Создайте диаграмму видов деятельности
2. В этом бизнес-процессе участвуют четыре объекта: клиент, менеджер, бухгалтер и кладовщик. Создайте дорожки (swimlanes) для каждого из объектов. Каждая из дорожек ответственна за выполнение определенных действий тем объектом, с которым она проассоциирована.
3. Далее необходимо расположить на диаграмме все деятельности/действия, которые выполняются тем или иным объектом.
4. Результат: диаграмма, представленная на рисунке:



5. Сохраните диаграмму.

Эта диаграмма также может оказаться опорной для построения диаграммы классов.

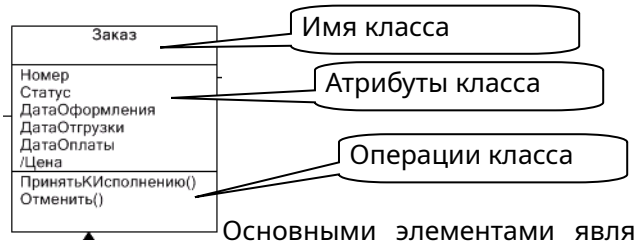
Построение диаграммы классов (Class Diagram)

После того, как мы определились с функциональными требованиями к системе и её границами, начнём анализировать предметную область с целью построения диаграммы классов.

Краткие теоретические сведения о диаграммах классов

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами. Вид и интерпретация диаграммы классов существенно зависит от точки зрения (уровня абстракции): классы могут представлять сущности предметной области (в процессе анализа) или элементы программной системы (в процессах проектирования и реализации).

Основные элементы диаграммы классов

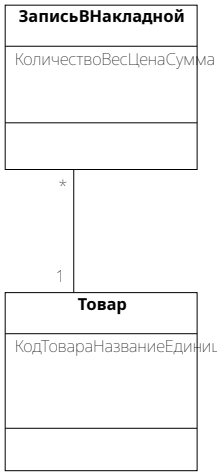


Основными элементами являются классы и связи между ними. Классы характеризуются при помощи *атрибутов* и *операций.*

Атрибуты описывают свойства объектов класса. Большинство объектов в классе получают свою индивидуальность из-за различий в их атрибутах и взаимосвязи с другими объектами. Однако, возможны объекты с идентичными значениями атрибутов и взаимосвязей. Т.е. индивидуальность объектов определяется самим фактом их существования, а не различиями в их свойствах. Имя атрибута должно быть уникально в пределах класса. За именем атрибута может следовать его тип и значение по умолчанию.

Операция есть функция или преобразование. Операция может параметризоваться и возвращать значения.

Виды связей: ассоциация, агрегация и наследование.



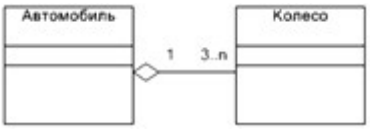
Ассоциация (association) – представляет собой отношения между экземплярами классов.

Каждый конец ассоциации обладает ***кратностью (синоним – мощностью, ориг. — multiplicity)***, которая показывает, сколько объектов, расположенных с соответствующего конца ассоциации, может участвовать в данном отношении. В примере на рисунке каждый Товар имеет сколь угодно Записей в накладной, но каждая Запись в накладной обязательно один Товар. В общем случае кратность может быть задана любым множеством.

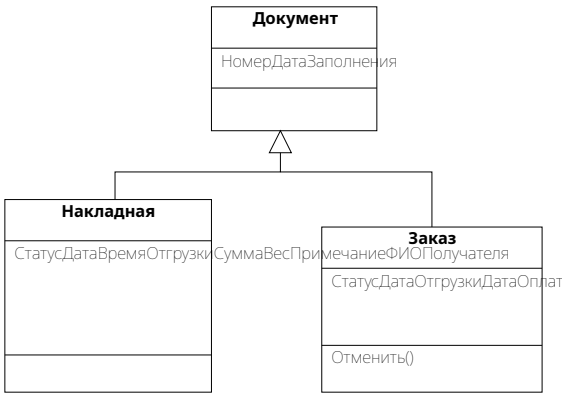
Ассоциации может быть присвоено имя. В качестве имени обычно выбирается глагол или глагольное словосочетание, сообщающие смысл и назначение связи.

Также, на концах ассоциации под кратностью может указываться имя роли, т.е. какую роль выполняют объекты, находящиеся с данного конца ассоциации.

Агрегация (aggregation) – это ассоциация типа «целое-часть». Агрегация в UML представляется виде прямой с ромбом на конце.



Ромб на связи указывает, какой класс является агрегирующим (т.е. «**состоящим из»**), — класс с противоположного конца — агрегированным (т.е. те самые «**части**»).



Композиция (composition) **–** это такая агрегация, где объекты-части не могут существовать сами по себе и уничтожаются при уничтожении объекта агрегирующего класса. Композиция изображается так же как ассоциация, только ромбик закрашен.

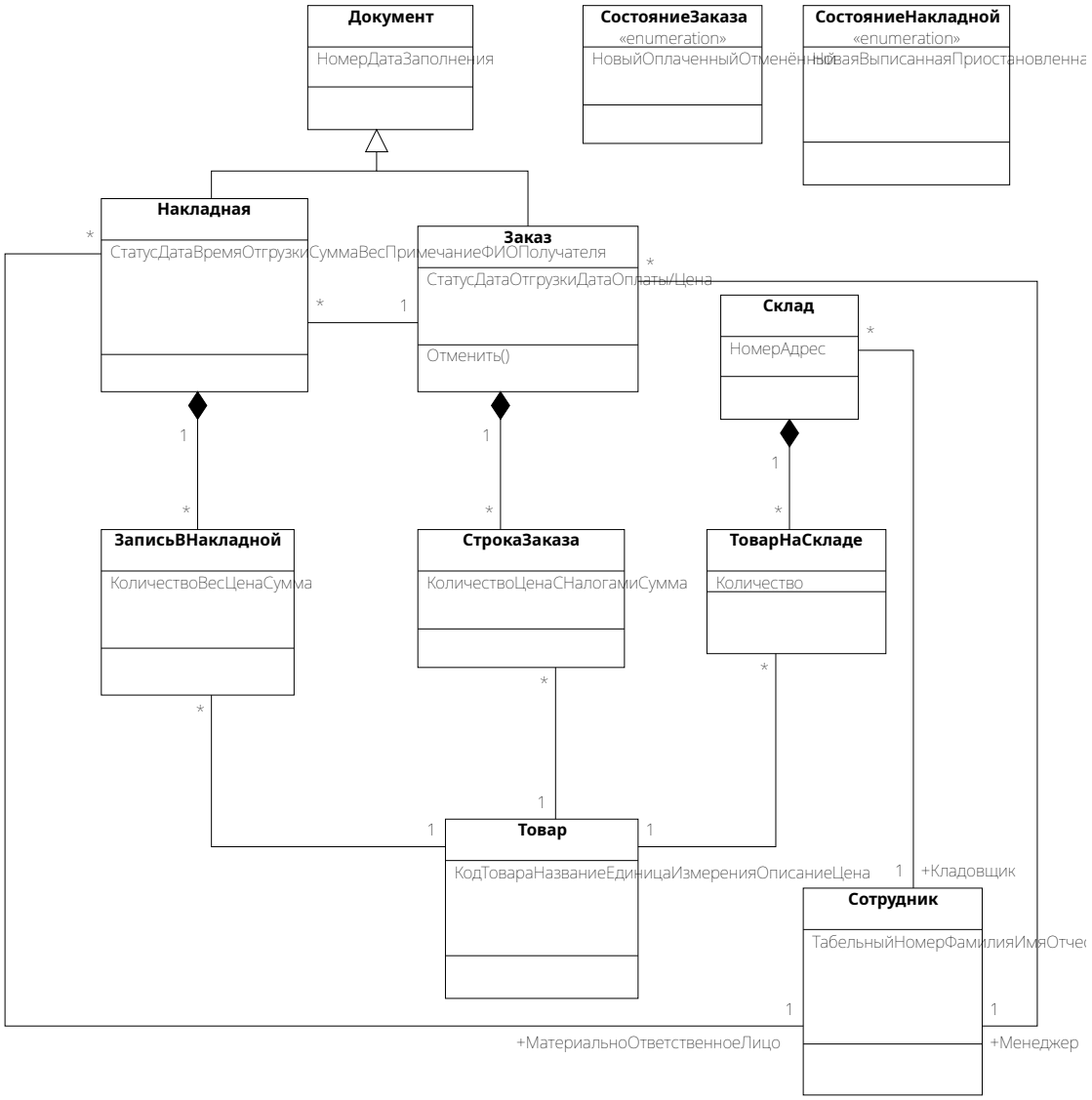
Важно понимать разницу между агрегацией и композицией: при агрегации объекты-части могут существовать сами по себе, а при композиции — нет. Пример агрегации: автомобиль—колесо, пример композиции: дом —комната.

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

структурой ряда других классов. При создании производного класса на основе базового (одного или нескольких) возникает иерархия наследования. Реализация принципов наследования является ключевой предпосылкой возможности повторного использования кода, поскольку это основной инструмент достижения полиморфизма.

Порядок построения диаграммы классов

1. Создаём новую диаграмму с именем «Сущности».
2. Проанализируйте предметную область и постройте диаграмму классов. У вас должно получиться нечто подобное:



Давайте поподробнее рассмотрим эту диаграмму. Основной сущностью в нашей системе будет являться товар. Как мы знаем из задания на проектирование, товар хранится на складе. Но понятия товара как некоего описания и товара лежащего непосредственно на складе отличаются друг от друга. Товар, лежащий на складе, кроме того что связан со складом отношением композиции (агрегация не совсем подходит, поскольку в данной системе товар является товаром, пока он не покинет склад), ещё характеризуется количеством. Аналогично следует рассуждать и при рассмотрении отношения Товара и Заказа, Товара и Накладной. В связи с тем, что Заказ и Накладная в сущности являются документами и имеют сходные атрибуты, они были объединены с помощью общего класса-предка Документ. Также заметьте, что тут есть два класса со стереотипом «Enumeration» (перечисление). Стереотип можно установить из контекстного меню для класса.

3. Сохраните диаграмму

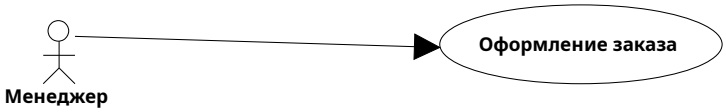
Построение диаграмм взаимодействия (Interaction diagram)

Краткие теоретические сведения

В нотации UML взаимодействие элементов рассматривается в информационном аспекте их коммуникации. Другими словами, взаимодействующие объекты обмениваются между собой некоторой информацией. При этом информация представляет собой законченные сообщения.

Для описания взаимодействия объектов, участвующих в некотором прецеденте, используются сценарии. ***Сценарий*** – это экземпляр прецедента, определяющий один из возможных потоков событий данного прецедента. Сам прецедент представляет собой переплетение сценариев – как основных, представляющих нормальное течение событий, так и вспомогательных, определяющих логику функционирования системы в ситуациях вида «что произойдет, если…». На ранних стадиях проектирования системы, как правило, ограничиваются рассмотрением основного сценария для каждого выявленного прецедента.

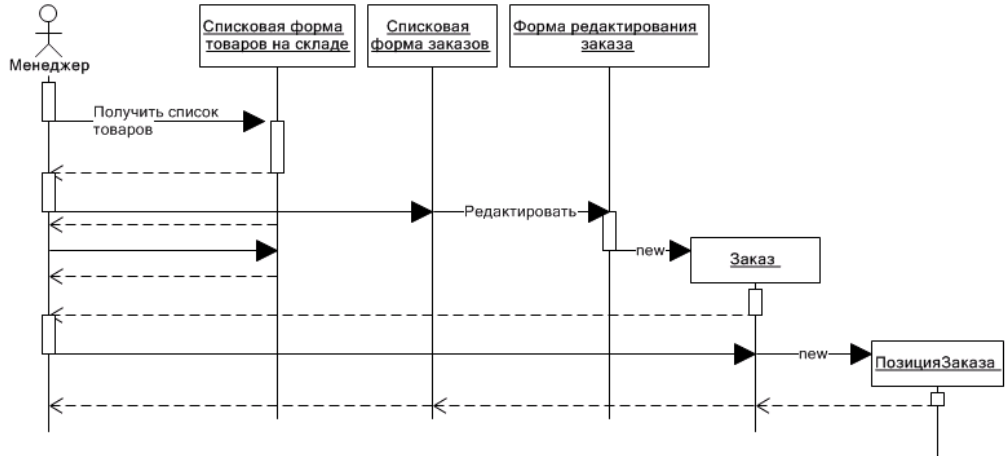
Для примера создадим реализацию прецедента «Оформление заказа»:



Построим для данной реализации **диаграмму последовательностей**, которая будет описывать сценарий размещения нового заказа.

Порядок построения диаграммы последовательностей (Sequence Diagram)

1. Создайте диаграмму последовательностей «Создание нового заказа»
2. На данной диаграмме отразим взаимодействие объектов классов: «Менеджер», «Списковая форма товаров на складе», «Списковая форма заказов», «Форма редактирования заказов», «Заказ», «Позиция Заказов». Построенная диаграмма представлена на рисунке:



3. Сохраните диаграмму.

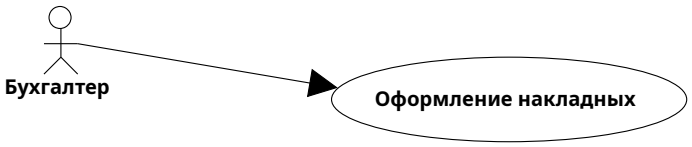
Диаграммы последовательностей не только отображают взаимодействие объектов, но и позволяют определить/отыскать операции, которые должны иметь те или иные классы.

Краткие теоретические сведения

Диаграмма сотрудничества представляет альтернативный способ описания взаимодействия объектов и акцентирует внимание в первую очередь на организации

объектов. Сообщения между объектами обозначаются стрелками, однако, их временная последовательность определяется нумерацией стрелок.

Создадим реализацию прецедента «Оформление накладных» - построим диаграмму кооперации для сценария «Печать накладной».



Построение диаграммы сотрудничества (Collaboration Diagram)

1. Создайте диаграмму сотрудничества
2. Разместите следующие элементы:

3. Сохраните диаграмму.

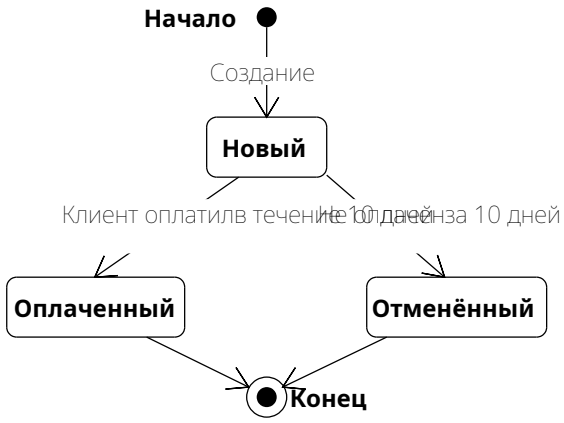
Построение диаграмм состояний (Statechart Diagram)

Диаграмма состояний определяет последовательность состояний объекта, вызванных последовательностью событий.

Порядок построения диаграммы

1. Создайте диаграмму состояний для объектов класса «Заказ».
2. Из спецификации прецедентов следует, что заказ может быть в трех состояниях:
   - «Новый»;
   - «Оплаченный»;
   - «Отмененный».

В состояние «Новый» заказ попадает сразу после своего создания и находится в нем до момента перевода его менеджером в состояние «Оплаченный». Событием к переходу является поступление денег в кассу. Условие перехода – оплата должна производиться не позднее 10 дней со дня оформления заказа. В случае если оплата не производится в течение отведенных 10 дней или производится позже, заказ переходит в состояние «Отмененный». Соответствующая диаграмма состояний представлена на рисунке:



3. Сохраните диаграмму.
4. Создайте диаграмму состояний для объектов класса Накладная
5. Рассмотрим построение диаграммы состояний для товарно-транспортной накладной. Все вновь созданные накладные попадают в состояние «Новая». После печати накладной она переходит в состояние «Выписанная». В этот момент электронная накладная становится доступной кладовщику на складе, и он начинает сборку заказа. По окончании сборки кладовщик переводит накладную в состояние «Готовая». Если по каким-то причинам на складе не оказалось нужного товара (брак в партии, просрочка поставщика и т.п.), что делает невозможным сборку заказа, накладная переходит в состояние «Приостановленная». После того как товар отгружен клиенту, накладная переходит в состояние «Отгруженная». Диаграмма состояний для накладной изображена на рисунке:

6. Сохраните диаграмму.

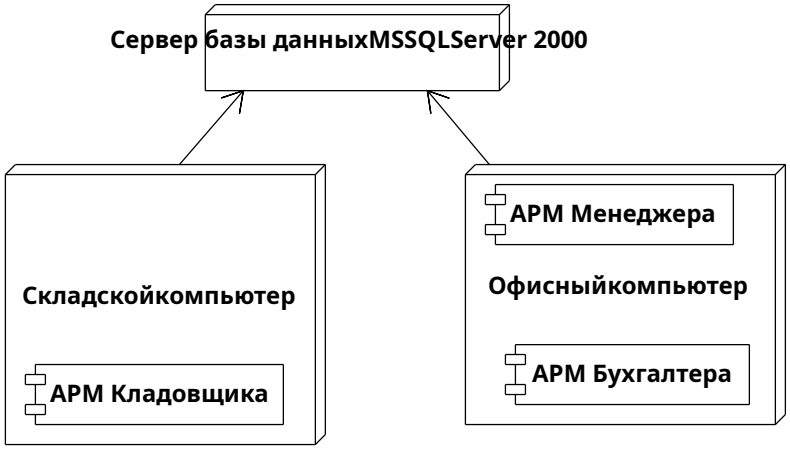
Диаграмма развёртывания (Deployment diagram)

Диаграмма развертывания является физической диаграммой в языке UML. Она отображает физические взаимосвязи между программными и аппаратным компонентами проектируемой системы.

Порядок создания диаграммы

1. Создайте диаграмму развёртывания.

2. Расположите элементы как показано на рисунке:



3. Сохраните диаграмму.

Контрольные вопросы:

1. Для чего применяются CASE-системы?
2. Какие виды UML-диаграмм используются для описания будущей системы?
3. Приведите примеры различных видов связей на диаграмме классов.
4. Для чего нужны плавательные дорожки?

Лабораторная работа №2

Прототипирование и создание приложения

Цель работы: научиться создавать готовые приложения с использованием CASEBERRY.

Введение

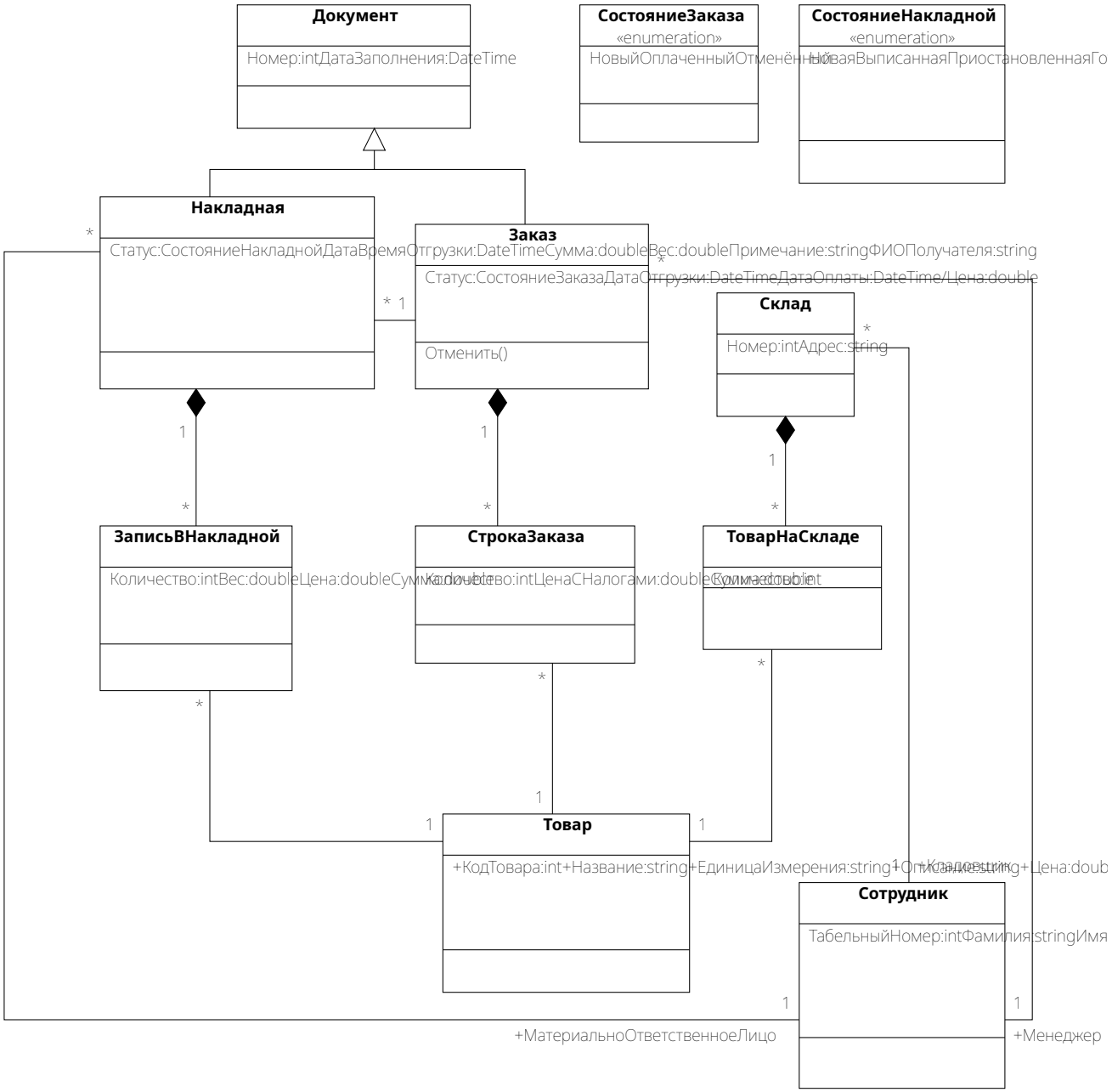
Целью данной лабораторной работы является сгенерированный и откомпилированный код, то есть готовое приложение. Но для того, чтобы модуль генерации кода мог сгенерировать это приложение, нужно создать представления, классы форм для редактирования, классы форм для списков, класс приложения, и сделать некоторые настройки. Чтобы облегчить этот процесс, в CASEBERRY используется быстрое прототипирование, которое выполняет эти действия некоторым предопределённым образом. Несмотря на то, что результат не всегда превосходный, но зато мы быстро получаем работающее приложение, которое может использоваться как прототип при обсуждении с заказчиком. И, самое главное, прототип получен почти «мгновенно», без ощутимых затрат времени. Впоследствии, созданные прототипизацией представления, классы форм для редактирования, классы форм для списков, класс приложения, можно модифицировать, приближая систему, тем самым, к требуемой функциональности.

Уточнение модели

Для того чтобы можно было выполнить быструю прототипизацию, нужна диаграмма классов. В ней должны быть указаны все необходимые атрибуты и их типы, а также правильно описаны операции.

Порядок выполнения данного этапа

1. Откройте диаграмму классов «Сущности», которая была сделана в первой лабораторной работе.
2. Укажите типы атрибутов классов, как это указано на следующей диаграмме:



Примечание:

1) типы атрибутов можно проставлять и при первом рассмотрении системы, но зачастую на первых этапах они упускаются в связи с неопределённостью, или они просто ещё не нужны. Главное, чтобы перед прототипированием типы были проставлены.

2) Чтобы убедиться в корректности введённых атрибутов, можно просто попробовать посмотреть свойства данного класса (контекстное меню (п. к. м.) -> Редактировать свойства). Если формат атрибутов и операций удовлетворяет систему, то отобразятся свойства, иначе появится предупреждение. (Имена типов не проверяются.)

3) При генерации кода на диаграмме классов не должно быть связей типа агрегация (может быть только ассоциация и композиция) и связей с множественностью 1 к 1.

3. Сохраните диаграмму.

Создание прототипа

Перед созданием прототипа ещё раз убедитесь в том, что в модели всё верно, потому что исправление модели после создания прототипа будет более трудоёмким. (Нужно будет удалить автоматически созданную диаграмму и представления в классах и повторить быстрое прототипирование).

Порядок создания прототипа

1. Перейдите на созданную вами систему:

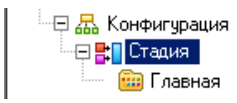
2. В меню появится пункт «CSharp», а там «Создать прототип», который и нужно выбрать.
3. В появившемся окне укажите имя приложения (можно указать любое, главное, чтобы оно не совпало с именем уже имеющегося класса), например АСУ_Склад. В качестве префикса можно указать букву «С». После этого нажмите на «ОК».
4. Если всё сделано верно, прототип будет успешно создан и появится новая диаграмма классов АСУ_Склад (имя диаграммы совпадает с именем приложения).

Настройка приложения

1. Откройте диаграмму с приложением (АСУ_Склад).
2. Выберите «Редактировать свойства» из контекстного меню для класса АСУ_Склад.
3. На вкладке «Конфигурация» в появившемся окне свойств укажите имя сервера и имя базы данных. (Их должен выдать преподаватель.) Если рассмотреть строку соединения поподробнее «SERVER=server;Trusted_connection=yes;DATABASE=database;», то будет видно, что, вместо «server» надо указать имя конкретного сервера, а вместо database имя конкретной базы данных, с которой будет работать ваше приложение. Скопируйте в буфер обмена строку соединения – она понадобиться вам на следующем этапе.
4. Сохраните настройки и закройте окно настроек и диаграммы.

Настройка генератора скриптов для БД

1. Перейдите на стадию:



2. В меню появится пункт «MS SQL Server direct generator». Выбираем «Редактировать»
3. В появившемся окне в поле «Строка соединения» указываем то же самое, что в настройках приложения было указано (должно быть в буфере обмена).
4. Сохраняем и закрываем окно.

Настройка модуля кодогенерации (CSharp)

1. Перейдите на стадию (как в предыдущем разделе)
2. Меню -> CSharp -> Редактировать
3. В поле Product указываем имя программного продукта «АСУСклад», также можно настроить путь до каталога, в который будут помещены сгенерированные проекты.
4. Сохраните и закройте окно.

Создание структуры данных

1. Перейдите на стадию
2. MS SQL Server direct generator -> Привести в соответствие БД SQL Server
3. Если такой базы нет, то она будет создана. Также будет сгенерирован скрипт, который представляет собой набор SQL-DDL-команд, применив которые, вы получите структуру базы данных, соответствующую модели.

Сборка

Краткие теоретические сведения

При первоначальной генерации-компиляции важно соблюдать порядок и поэтапность генерации-компиляции.

Правильный порядок:

1. Сгенерировать сборки классов данных (XXXXX(Objects));
2. Компилируются сборки классов данных;
3. Генерируются формы (XXXXX(Forms)), приложения, сборки с настройками рабочего стола (XXXXX(Desktop Customizers));
4. Компилируются формы;
5. Компилируются сборки с настройками рабочего стола;
6. Компилируются приложения.

Существует функция, весьма удобная в использовании при первоначальной генерации-компиляции, автоматически следующая вышеуказанному порядку - сборка системы.

Порядок выполнения сборки

1. Перейдите на стадию.
2. CSharp -> Собрать
3. Если не произойдёт ошибок при генерации и компиляции, то появится диалог с предложением запустить приложение.
4. Запустите приложение, ознакомьтесь с его работой.

Контрольные вопросы

1. Где указывается строка соединения с сервером?

2. Найдите взаимосвязь между связями на диаграмме классов и элементами управления в приложении.
3. Какая диаграмма используется при кодогенерации?
4. В какой последовательности должны генерироваться и компилироваться объекты данных и формы? Как Вы думаете, почему?