Министерство образования республики Беларусь

Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

**Институт информационных технологий**

Специальность «Программное обеспечение информационных технологий»\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**КОНТРОЛЬНАЯ РАБОТА**

По курсу: «Технология разработки программного обеспечения»

На тему: «Учёт товаров на складе»

Студент-заочник\_2\_ курса

Группы №\_581072\_\_\_\_\_\_\_\_\_\_\_

ФИО\_Мальчик Дмитрий\_\_\_\_\_\_

Александрович\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Адрес\_г. Минск, ул. Васнецова,

д. 5, кв. 80\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Тел.\_+375292708981\_\_\_\_\_\_\_\_\_

Минск, 2017

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 3

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 4

1.1 Постановка задачи 4

1.2 Анализ требований к автоматизированной системе 4

1.3. Обзор существующих аналогов 5

1.4 Описание и обоснование выбранного языка программирования 6

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА. 9

3 ЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ 14

3.1 Диаграмма вариантов использования 14

3.2 Диаграмма классов 15

3.3 Диаграмма деятельности 16

3.4 Диаграмма коопераций 17

3.5 Диаграмма последовательности 18

3.6 Диаграмма развёртывания 19

3.7 Диаграмма состояний 20

3.8 Диаграмма компонентов 21

СПИСОК ИСПОЛЬЗОВАНЫХ ИСТОЧНИКОВ 23

Генерация кода 24

# Введение

В современном обществе все предприятия имеют дело с потоками различной информации, которую необходимо оперативно обрабатывать. Объем информации зависит в основном от размера предприятия и вида деятельности: чем больше предприятие, тем больше количество и уровень сложности обрабатываемой информации. Немалую помощь здесь оказывают современные информационные технологии. Компьютерная информационная система позволяет вести учёт на предприятии быстро и точно, предоставляет широкие возможности анализа, избавляет от огромного количества бумаги.

Основой для учета, контроля и планирования служат всевозможные картотеки, регистрационные журналы, списки и т.д. Они постепенно накапливаются и обновляются. При большом объеме информации поиск и обобщение необходимых сведений, осуществляемых вручную, представляют собой довольно трудоемкий процесс.

С появлением ЭВМ и использованием их для обработки информации появилась возможность автоматизировать решение многих информационно-справочных и расчетных задач.

Предметная область информационной системы - это материальная система или система, характеризующая элементы материального мира, информация о которой хранится и обрабатывается. Предметная область рассматривается как некоторая совокупность реальных объектов и связей между ними.

Склад готовой продукции не занимается никакой коммерческой деятельностью, а только осуществляет процедуру хранения продукции для сторонних лиц заинтересованных в этом. Склад должен выполнять следующие функции: прием, учет, хранение и отгрузка готовой продукции, приемка готовой продукции, рассортировка, комплектация потребителям, определение потребности в транспортных средствах, механизированных погрузочных средствах, таре и рабочей силе для отгрузки продукции, согласование планов и условий поставок продукции с основного производства и по договорам со сторонними организациями, организация приемки продукции сторонними организациями, координация деятельности по закупке и продаже продукции с наличием свободных складских площадей, подготовка отчетов об объемах продукции, а также участие в рассмотрении поступающих на предприятие претензий.

Затем склад готовой продукции должен предоставить создание условий для сохранности продукции, находящейся на временном хранении, организацию рационального хранения, внутренней транспортировки, упаковки и подготовки продукции к отправке, обеспечение сохранности продукции, обеспечение высокого уровня механизации и автоматизации транспортно-складских операций, применения компьютерных систем и нормативных условий организации и охраны труда.

# 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## **Постановка задачи**

В данной контрольной работе необходимо реализовать программное средство «Учёт товаров на складе». Программа должна выполнять следующие функции:

* заполнение накладных;
* заполнение справочников;
* формирование отчетов;
* вывод отчетов на печать.

**1.2 Анализ требований к автоматизированной системе**

Процесс проектирования архитектуры программного обес­печения состоит в проектировании структуры всех его ком­понент, функционально связанных с решаемой задачей, включая сопряжения между ними и требования к ним.

Архитектура программного обеспечения в традиционном смысле включает определение всех модулей программ, их иерархии и сопряжения между ними и данными.

Если разрабатывается отдельная программа, исходными дан­ными для этого процесса будут детальные внешние специ­фикации.

Во время разработки архитектуры программного обеспечения выполняется его модульно-иерархическое построение.

Модуль — это замкнутая программа, которую можно вызвать из любого другого модуля в программе и можно отдельно ком­пилировать.

Реализованы следующие функции программного средства:

Функции, выполняемые программой:

* хранение информации об объектах, работниках, организации и др.;
* возможность добавления информации о новых объектах, организации и др.;
* возможность редактирования информации;
* возможность удаления информации;
* возможность формирования печатной формы.

Входной информацией в разрабатываемом программном обеспечении являются сведения о товарах на складе, количестве, поступлении новых товаров.

При работе пользователя с разрабатываемой информационной системой не должно возникать проблем, так как система обладает понятным программным интерфейсом.

**1.3 Обзор существующих аналогов**

Система 1С: Предприятие 8.2 имеет в своей основе ряд механизмов, определяющих концепцию создания прикладных решений. Наличие этих механизмов позволяет максимально соотнести технологические возможности с бизнес-схемой разработки и внедрения прикладных решений.

В качестве ключевых моментов можно выделить изоляцию разработчика от технологических подробностей, алгоритмическое программирование только бизнес-логики приложения, использование собственной модели базы данных и масштабируемость прикладных решений без их доработки.

Состав прикладных механизмов 1С: Предприятия ориентирован на решение задач автоматизации учета и управления предприятием. Использование проблемно-ориентированных объектов позволяет разработчику решать самый широкий круг задач складского, бухгалтерского, управленческого учета, расчета зарплаты, анализа данных и управления на уровне бизнес-процессов.

Система 1С: Предприятие 8.2 является открытой системой. Предоставляется возможность для интеграции практически с любыми внешними программами и оборудованием на основе общепризнанных открытых стандартов и протоколов передачи данных.

В системе 1С: Предприятие 8.2 имеется целый набор средств, с помощью которых можно:

* создавать, обрабатывать и обмениваться данными различных форматов;
* осуществлять доступ ко всем объектам системы 1С: Предприятие 8, реализующим ее функциональные возможности;
* поддерживать различные протоколы обмена;
* поддерживать стандарты взаимодействия с другими подсистемами;
* создавать собственные интернет-решения.

Преимущества 1C: Предприятие 8.2

**Хранение сведений:**

В базе данных можно хранить произвольную информацию в разрезе нескольких измерений. Для этого введен новый объект конфигурации - Регистр сведений. Он поддерживает периодичность, так что информация может быть развернута по времени или позиции документа.

**Экономическая и аналитическая отчетность:**

а) Платформа содержит набор готовых оформлений. Кроме того, пользователь может самостоятельно создавать собственные варианты оформления.

б) Появился новый мощный инструмент интерактивного анализа данных – Сводная таблица. Она может быть размещена в табличном документе и позволяет интерактивно настраивать представление итоговых данных.

Недостатки 1С: Предприятие 8.2

* Высокие системные требования. Даже пустая база данных весит около 200 мегабайт. Поэтому и компьютер должен быть соответствующий – от 512 мегабайт оперативной памяти (а для комфортной работы лучше 1024 мегабайта), процессор от 1500 мегагерц.
* Низкая производительность платформы. Даже если установить 1С: Предприятие 8 на очень шустрый компьютер, то высокой скорости работы добиться будет проблематично.
* Для того чтобы работать с 8.2, необходимо обучение персонала. Для того чтобы пользователь смог разобраться в механизме стандартных отчётов, ему надо иметь хотя бы знания начального уровня о том, как хранятся данные в 1С, о том, что такое регистры, о том, как проводятся документы, что при этом происходит, т.е. для работы с 1С версии 8 необходим более квалифицированный персонал.
* В 1С версии 8.2 больше возможностей, отчётов, вариантов работы.
* 1С, версия 8.2 – изменяется и обновляется регулярно, несколько раз в месяц, что по большому счёту сказывается на стоимости владения и обслуживания.

# 1.4 Описание и обоснование выбранного языка программирования

Java - объектно-ориентированный язык программирования, разрабатываемый компанией Sun Microsystems с 1991 года и официально выпущенный 23 мая 1995 года. Изначально новый язык программирования назывался Oak (James Gosling) и разрабатывался для бытовой электроники, но впоследствии был переименован в Java и стал использоваться для написания апплетов, приложений и серверного программного обеспечения

Отличительной особенностью Java в сравнении с другими языками программирования общего назначения является обеспечение высокой продуктивности программирования, нежели производительность работы приложения или эффективность использования им памяти.

В Java используются практически идентичные соглашения для объявления переменных, передачи параметров, операторов и для управления потоком выполнением кода. В Java добавлены все хорошие черты C++.

Три ключевых элемента объединились в технологии языка Java:

1) Java предоставляет для широкого использования свои апплеты (applets) — небольшие, надежные, динамичные, не зависящие от платформы активные сетевые приложения, встраиваемые в страницы Web. Апплеты Java могут настраиваться и распространяться потребителям с такой же легкостью, как любые документы HTML.

2) Java высвобождает мощь объектно-ориентированной разработки приложений, сочетая простой и знакомый синтаксис с надежной и удобной в работе средой разработки. Это позволяет широкому кругу программистов быстро создавать новые программы и новые апплеты.

3) Java предоставляет программисту богатый набор классов объектов для ясного абстрагирования многих системных функций, используемых при работе с окнами, сетью и для ввода-вывода. Ключевая черта этих классов заключается в том, что они обеспечивают создание независимых от используемой платформы абстракций для широкого спектра системных интерфейсов.

Огромное преимущество Java заключается в том, что на этом языке можно создавать приложения, способные работать на различных платформах. К сети Internet подключены компьютеры самых разных типов - Pentium PC, Macintosh, рабочие станции Sun и так далее. Даже в рамках компьютеров, созданных на базе процессоров Intel, существует несколько платформ, например, Microsoft Windows версии 3.1, Windows 95, Windows NT, OS/2, Solaris, различные разновидности операционной системы UNIX с графической оболочкой X­Windows. Между тем, создавая сервер Web в сети Internet, хотелось бы, чтобы им могло пользоваться как можно большее число людей. В этом случае выручат приложения Java, предназначенные для работы на различных платформах и не зависящие от конкретного типа процессора и операционной системы.

Программы, составленные на языке программирования Java, можно разделить по своему назначению на две большие группы.

К первой группе относятся приложения Java, предназначенные для автономной работы под управлением специальной интерпретирующей машины Java. Реализации этой машины созданы для всех основных компьютерных платформ.

Вторая группа - это так называемые аплеты (applets). Аплеты представляют собой разновидность приложений Java, которые интерпретируются виртуальной машиной Java, встроенной практически во все современные браузеры.

Приложения, относящиеся к первой группе - это обычные автономные программы. Так как они не содержат машинного кода и работают под управлением специального интерпретатора, их производительность заметно ниже, чем у обычных программ, составленных, например, на языке программирования C++. Программы Java без перетрансляции способны работать на любой платформе, что само по себе имеет большое значение в плане разработок для Internet.

Аплеты Java встраиваются в документы HTML, хранящиеся на сервере Web. С помощью аплетов вы можете сделать страницы сервера Web динамичными и интерактивными. Аплеты позволяют выполнять сложную локальную обработку данных, полученных от сервера Web или введенных пользователем с клавиатуры. Из соображений безопасности аплеты (в отличие от обычных приложений Java) не имеют никакого доступа к файловой системе локального компьютера. Все данные для обработки они могут получить только от сервера Web. Более сложную обработку данных можно выполнять, организовав взаимодействие между аплетами и расширениями сервера Web - приложениями CGI и ISAPI.

Для повышения производительности приложений Java в современных браузерах используется компиляция "на лету"- Just-In-Time compilation (JIT). При первой загрузке аплета его код транслируется в обычную исполнимую программу, которая сохраняется на диске и запускается. В результате общая скорость выполнения аплета Java увеличивается в несколько раз.

Язык Java является объектно-ориентированным и поставляется с достаточно объемной библиотекой классов. Так же как и библиотеки классов систем разработки приложений на языке C++, библиотеки классов Java значительно упрощают разработку приложений, представляя в распоряжение программиста мощные средства решения распространенных задач. Поэтому программист может больше внимания уделить решению прикладных задач.

Язык Java специально ориентирован на самые передовые технологии, связанные с сетью Internet. Растущая популярность Internet и, в особенности, серверов Web, создает для программистов новые возможности для реализации своих способностей.

**2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

В работе проводится моделирование с использование IDEF0 (BPWin), UML (Rational Rose 2000), IDEF1x (ErWin).

1. Важная роль отводится процессу функционального проектирования.

Для регламентирования создания функциональных моделей ПС предназначен стандарт IDEF0 (Integrated Definition Function Modeling), который и реализован в пакете BpWin.

В основе IDEF0 лежит понятие блока, который реализует некую конкретную функцию. Четыре стороны блока имеют разное назначение. Слева отображаются входные данные (исходные данные). Справа – выходные данные (результат выполнения функции). Сверху – управление (управляющие воздействия на функцию). Снизу – механизм (посредством чего реализуется данная функция).

Функция – это управляемое действие над входными данными, результатом которого являются выходные данные, при этом используется некий механизм. Взаимодействие между функциями отображается в виде стрелок. Иногда стороны блока называют направлениями, а стрелки потоками. Стрелки можно подписывать. Подписи связываются с конкретной стрелкой при помощи зигзага.

В основе IDEF0 лежит три базовых принципа:

1) принцип функциональной декомпозиции – любая функция может быть разбита (декомпозирована) на более простые функции (более понятен термин детализация);

2) принцип ограничения сложности – количество блоков на диаграмме должно быть не менее двух, но не более шести (условие удобочитаемости);

3) принцип контекста – моделирование делового процесса начинается с построения контекстной диаграммы, на которой отображается только один блок – главная функция моделирующей системы, ограничивающая область границы моделирующей системы (регламентирует начальный этап построения модели).

Под субъектом понимается сама система, при этом необходимо точно установить, что входит в систему, а что является внешним воздействием на систему. Т. е. первоначально нужно определить область моделирования. Описание области как системы в целом, так и ее компонентов является основой построения модели.

IDEF1X является методом для разработки реляционных баз данных и использует условный синтаксис, специально разработанный для удобного построения концептуальной схемы. Концептуальной схемой мы называем универсальное представление структуры данных в рамках коммерческого предприятия, независимое от конечной реализации базы данных и аппаратной платформы. Будучи статическим методом разработки, IDEF1X изначально не предназначен для динамического анализа по принципу "AS IS", тем не менее, он иногда применяется в этом качестве, как альтернатива методу IDEF1. Использование метода IDEF1X наиболее целесообразно для построения логической структуры базы данных после того, как все информационные ресурсы исследованы (скажем с помощью метода IDEF1) и решение о внедрении реляционной базы данных, как части корпоративной информационной системы, было принято.

Хотя терминология IDEF1X практически совпадает с терминологией IDEF1, существует ряд фундаментальных отличий в теоретических концепциях этих методологий. Сущность в IDEF1X описывает собой совокупность или набор экземпляров похожих по свойствам, но однозначно отличаемых друх от друга по одному или нескольким признакам. Каждый экземпляр является реализацией сущности. Таким образом, сущность в IDEF1X описывает конкретный набор экземпляров реального мира, в отличие от сущности в IDEF1, которая представляет собой абстрактный набор информационных отображений реального мира. Примером сущности IDEF1X может быть сущность "СОТРУДНИК", которая представляет собой всех сотрудников предприятия, а один из них, скажем, Иванов Петр Сергеевич, является конкретной реализацией этой сущности.

Сase-средство BPwin значительно облегчает задачу создания информационной системы, позволяя осуществить декомпозицию сложной системы на более простые с тем, чтобы каждая из них могла проектироваться независимо, и для понимания любого уровня проектирования достаточно было оперировать с информацией о немногих ее частях.

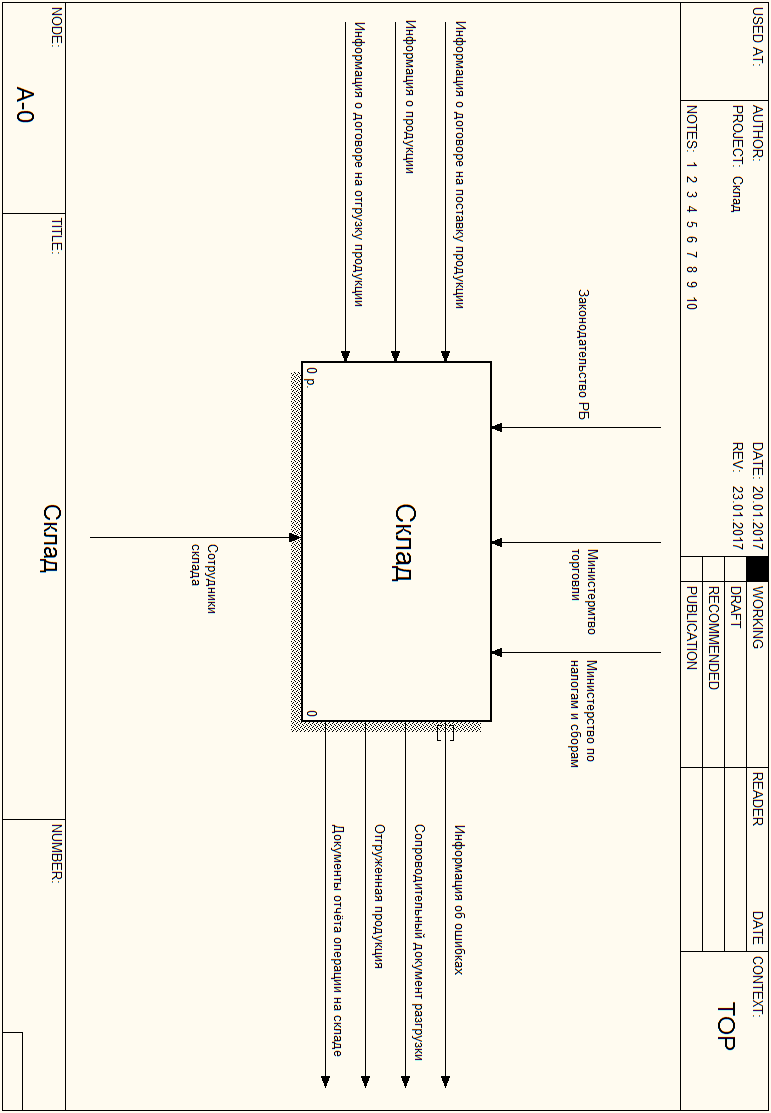


Рисунок 2.1– Контекстная диаграмма модели

В качестве исходной информации контекстной диаграммы используются данные о материалах и сотрудниках предприятия. Управляющими данными являются нормативные документы.

Результатом работы системы должны быть документы, созданные пользователем и отчёты с диаграммами. После построения контекстная диаграмма детализируется с помощью диаграммы декомпозиции первого уровня. На этой диаграмме отображаются функции системы, которые должны быть реализованы в рамках основной функции. Диаграмма, для которой выполнена декомпозиция, по отношению к детализирующим ее диаграммам называется родительской. Диаграмма декомпозиции по отношению к родительской называется дочерней.

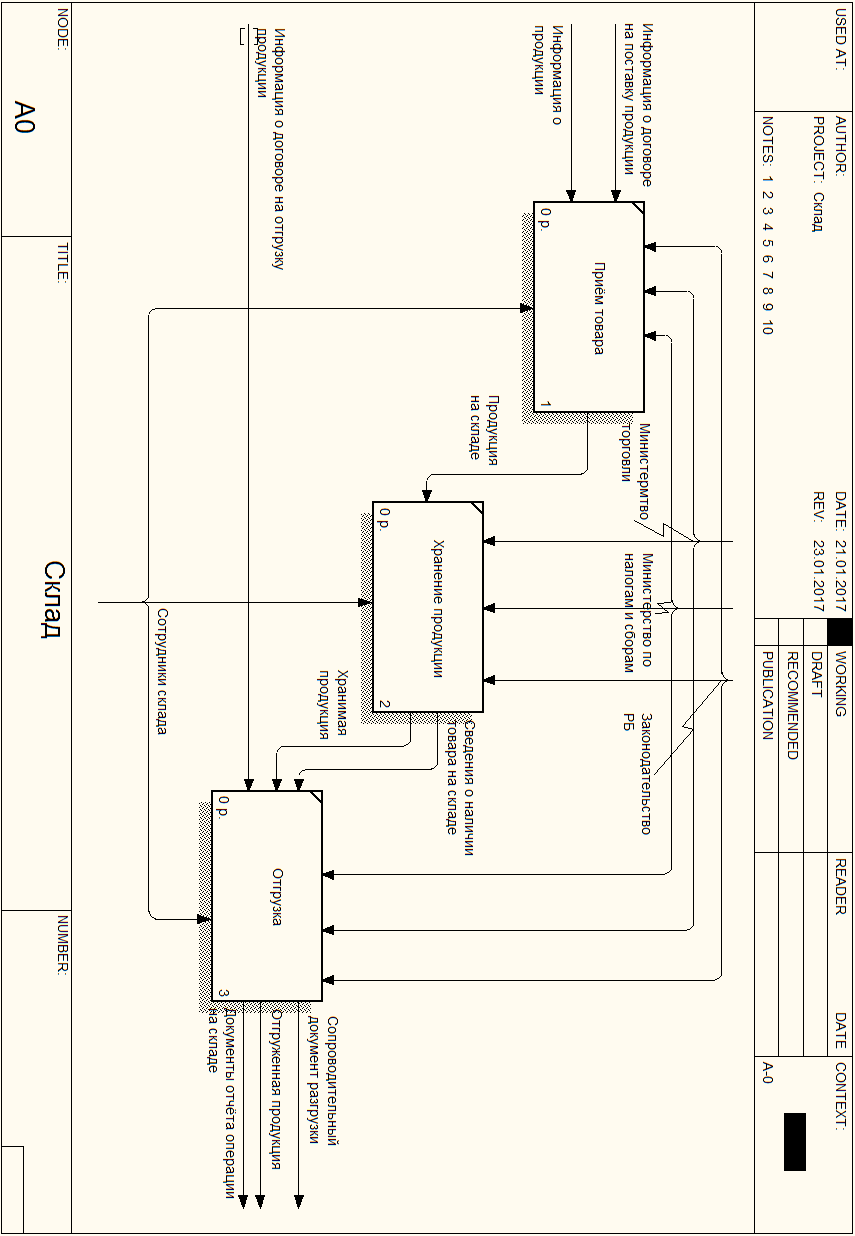


Рисунок 2.2 – Декомпозиция контекстной диаграммы функциональной модели

После построения диаграммы декомпозиции первого уровня для указанных на ней функций строятся отдельные диаграммы (диаграммы декомпозиции второго уровня). Затем процесс декомпозиции (построения диаграмм) продолжается до тех пор, пока дальнейшая детализация функций не теряет смысла. Для каждой атомарной функции, описывающей элементарную операцию (т. е. функции, не имеющей диаграмму декомпозиции), составляется подробная спецификация, определяющая ее особенности и алгоритм реализации. В качестве дополнения к спецификации могут использоваться блок-схемы алгоритмов. Таким образом, процесс функционального моделирования заключается в постепенном выстраивании иерархии функций.

**3 ЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ**

Основное назначение логического представления состоит в анализе структурных и функциональных отношений между элементами модели системы. Различные элементы логического представления, такие как классы, ассоциации, состояния, сообщения, не существуют материально или физически. Они лишь отражают наше понимание структуры физической системы или аспекты ее поведения.

### **3.1 Диаграмма вариантов использования**

Модель вариантов использования предназначается для определения требований к системе. Она включает в себя актеров, варианты использования и связи между ними. Для отображения этой модели язык UML предлагает использовать диаграммы Use Case (вариант использования) совместно с моделями State Diagram (диаграммы состояний) и Activity Diagram (диаграммы деятельности/активности). Последние используются для конкретизации вариантов использования системы

Разработка диаграммы вариантов использования преследует цели:

* определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы;
* сформулировать общие требования к функциональному поведению проектируемой системы;
* разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей;
* подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Диаграмма вариантов использования представлена на рисунке 3.1.

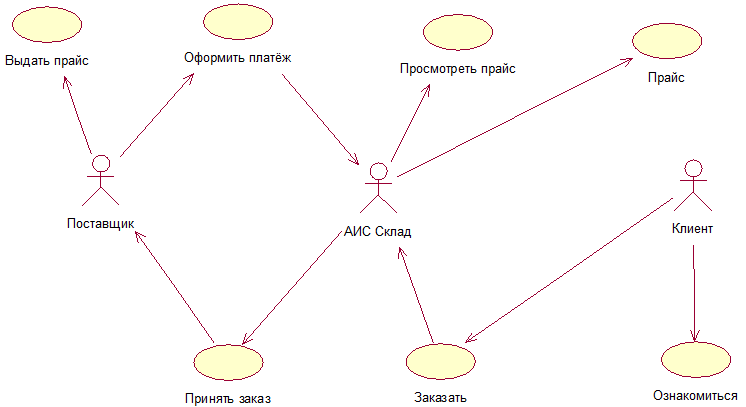


Рисунок 3.1 – Диаграмма вариантов использования

### **3.2 Диаграмма классов**

Диаграмма классов— диаграмма, демонстрирующая классы системы, их атрибуты, методы и взаимосвязи между ними. Входит в UML.

Существует два вида:

* Статический вид диаграммы рассматривает логические взаимосвязи классов между собой;
* Аналитический вид диаграммы рассматривает общий вид и взаимосвязи классов, входящих в систему.

Существуют разные точки зрения на построение диаграмм классов в зависимости от целей их применения:

* Концептуальная точка зрения — диаграмма классов описывает модель предметной области, в ней присутствуют только классы прикладных объектов;
* Точка зрения спецификации — диаграмма классов применяется при проектировании информационных систем;
* Точка зрения реализации — диаграмма классов содержит классы, используемые непосредственно в программном коде (при использовании объектно-ориентированных языков программирования).

Диаграмма классов представлена на рисунке 3.2.

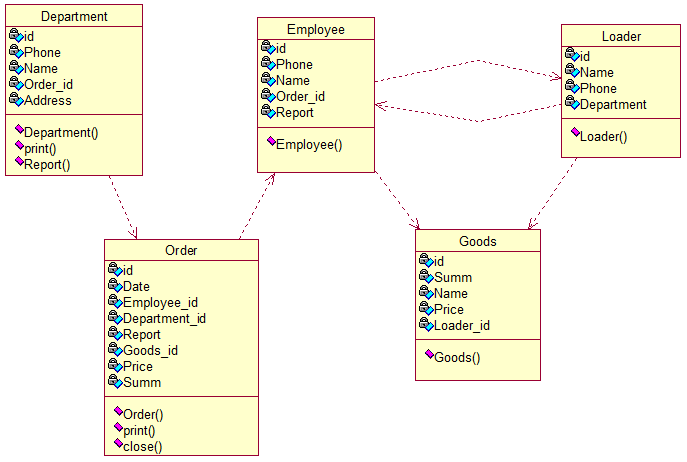


Рисунок 3.2 – Диаграмма классов

### **3.3 Диаграмма деятельности**

Диаграмма деятельности — UML-диаграмма, на которой показано разложение некоторой деятельности на её составные части. Под деятельностью (англ. activity) понимается спецификация исполняемого поведения в виде координированного последовательного и параллельного выполнения подчинённых элементов — вложенных видов деятельности и отдельных действий англ. action, соединённых между собой потоками, которые идут от выходов одного узла ко входам другого.

Диаграммы деятельности используются при моделировании бизнес-процессов, технологических процессов, последовательных и параллельных вычислений.

Диаграмма деятельности представлена на рисунке 3.3.

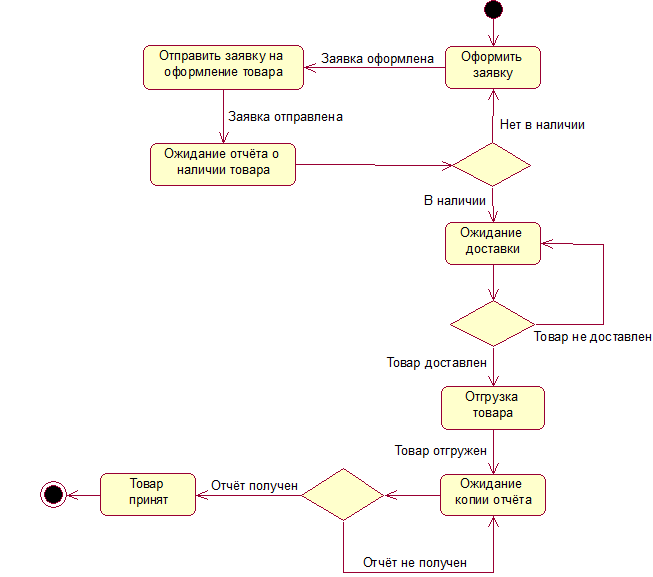


Рисунок 3.3 – Диаграмма деятельности

**3.4 Диаграмма коопераций**

Главная особенность диаграммы кооперации заключается в возможности графически представить не только последовательность взаимодействия, но и все структурные отношения между объектами, участвующими в этом взаимодействии.

Прежде всего, на диаграмме кооперации в виде прямоугольников изображаются участвующие во взаимодействии объекты, содержащие имя объекта, его класс и, возможно, значения атрибутов. Далее, как и на диаграмме классов, указываются ассоциации между объектами в виде различных соединительных линий. При этом можно явно указать имена ассоциации и ролей, которые играют объекты в данной ассоциации. Дополнительно могут быть изображены динамические связи - потоки сообщений. Они представляются также в виде соединительных линий между объектами, над которыми располагается стрелка с указанием направления, имени сообщения и порядкового номера в общей последовательности инициализации сообщений.

В отличие от диаграммы последовательности, на диаграмме кооперации изображаются только отношения между объектами, играющими определенные роли во взаимодействии. На этой диаграмме не указывается время в виде отдельного измерения. Поэтому последовательность взаимодействий и параллельных потоков может быть определена с помощью порядковых номеров. Следовательно, если необходимо явно специфицировать взаимосвязи между объектами в реальном времени, лучше это делать на диаграмме последовательности.

Диаграмма коопераций представлена на рисунке 3.4.

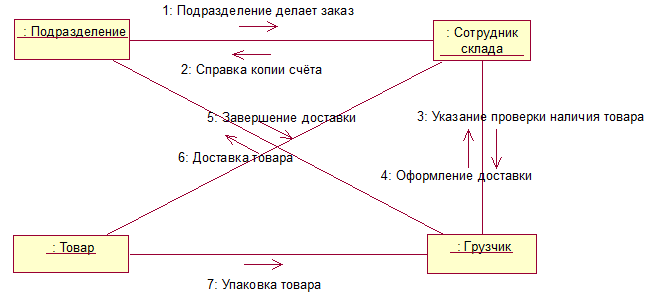


Рисунок 3.4 – Диаграмма коопераций

### **3.5 Диаграмма последовательности**

Диаграмма последовательности — диаграмма, на которой показано взаимодействие объектов, упорядоченное по времени, с отражением продолжительности обработки и последовательности их проявления. Используется в языке UML.

Основными элементами диаграммы последовательности являются обозначения объектов (прямоугольники с названиями объектов), вертикальные «линии жизни», отображающие течение времени, прямоугольники, отражающие деятельность объекта или исполнение им определенной функции (прямоугольники на пунктирной «линии жизни»), и стрелки, показывающие обмен сигналами или сообщениями между объектами.

На данной диаграмме объекты располагаются слева направо.

Диаграмма последовательности представлена на рисунке 3.5.



Рисунок 3.5 – Диаграмма последовательности

### **3.6 Диаграмма развёртывания**

Диаграмма развёртывания в UML моделирует физическое развертывание артефактов на узлах. Например, чтобы описать веб-сайт диаграмма развертывания должна показывать, какие аппаратные компоненты («узлы») существуют (например, веб-сервер, сервер базы данных, сервер приложения), какие программные компоненты («артефакты») работают на каждом узле (например, веб-приложение, база данных), и как различные части этого комплекса соединяются друг с другом (например, JDBC, REST, RMI).

Узлы представляются как прямоугольные параллелепипеды с артефактами, расположенными в них, изображенными в виде прямоугольников. Узлы могут иметь под узлы, которые представляются как вложенные прямоугольные параллелепипеды. Один узел диаграммы развертывания может концептуально представлять множество физических узлов, таких как кластер серверов баз данных.

Существует два типа узлов:

* узел устройства;
* узел среды выполнения.

Узлы устройств — это физические вычислительные ресурсы со своей памятью и сервисами для выполнения программного обеспечения, такие как обычные ПК, мобильные телефоны. Узел среды выполнения — это программный вычислительный ресурс, который работает внутри внешнего узла и который предоставляет собой сервис, выполняющий другие исполняемые программные элементы.

Диаграмма развёртывания представлена на рисунке 3.6.

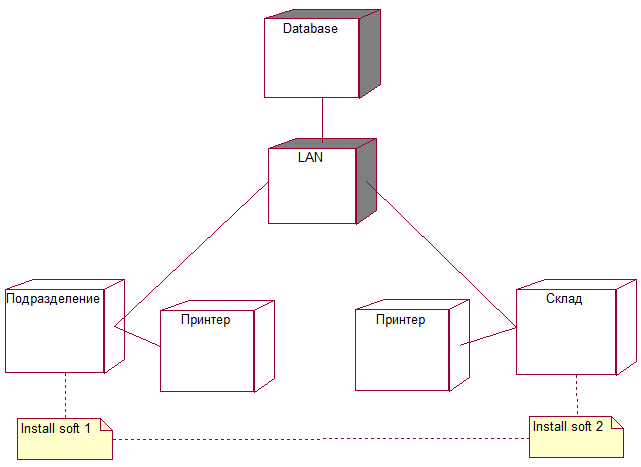


Рисунок 3.6 – Диаграмма развёртывания

### **3.7 Диаграмма состояний**

Диаграмма состояний — это, по существу, диаграмма состояний из теории автоматов со стандартизированными условными обозначениями, которая может определять множество систем от компьютерных программ до бизнес-процессов. Используются следующие условные обозначения:

* Круг, обозначающий начальное состояние.
* Окружность с маленьким кругом внутри, обозначающая конечное состояния (если есть).
* Скруглённый прямоугольник, обозначающий состояние. Верхушка прямоугольника содержит название состояния. В середине может быть горизонтальная линия, под которой записываются активности, происходящие в данном состоянии.
* Стрелка, обозначающая переход. Название события (если есть), вызывающего переход, отмечается рядом со стрелкой. Охраняющее выражение может быть добавлено перед «/» и заключено в квадратные скобки (название\_события[охраняющее\_выражение]), что значит, что это выражение должно быть истинным, чтобы переход имел место. Если при переходе производится какое-то действие, то оно добавляется после «/» (название\_события[охраняющее\_выражение]/действие).
* Толстая горизонтальная линия с либо множеством входящих линий и одной выходящей, либо одной входящей линией и множеством выходящих. Это обозначает объединение и разветвление соответственно.



Рисунок 3.7 – Диаграмма состояний

### **3.8 Диаграмма компонентов**

Диаграмма компонентов — статическая структурная диаграмма, показывает разбиение программной системы на структурные компоненты и связи (зависимости) между компонентами. В качестве физических компонентов могут выступать файлы, библиотеки, модули, исполняемые файлы, пакеты и т. п.

Компоненты связываются через зависимости, когда соединяется требуемый интерфейс одного компонента с имеющимся интерфейсом другого компонента. Таким образом иллюстрируются отношения клиент-источник между двумя компонентами.

Зависимость показывает, что один компонент предоставляет сервис, необходимый другому компоненту. Зависимость изображается стрелкой от интерфейса или порта клиента к импортируемому интерфейсу.

Когда диаграмма компонентов используется, чтобы показать внутреннюю структуру компонентов, предоставляемый и требуемый интерфейсы составного компонента могут делегироваться в соответствующие интерфейсы внутренних компонентов.

Делегация показывается связь внешнего контракта компонента с внутренней реализацией этого поведения внутренними компонентами.

Диаграмма компонентов представлена на рисунке 3.8.

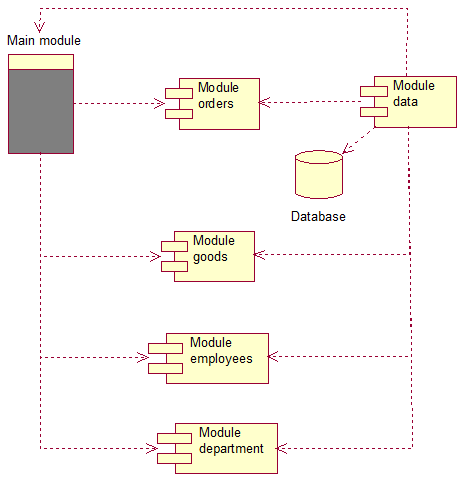


Рисунок 3.8 – Диаграмма компонентов

**СПИСОК ИСПОЛЬЗОВАНЫХ ИСТОЧНИКОВ**

1. ГОСТ 2.105-95 ЕСКД. Введ. 1996–07–01. – М.: Изд-во стандартов, 1996.
2. ГОСТ 19.104–78 – Единая система программной документации. Основные надписи.
3. Маклаков С.В. BPwin и ERwin: CASE - средства для разработки информационных систем.
4. Федотов Д.Э., Семенов Ю.Д., Чижик К.Н. Практикум для высших учебных заведений. CASE-технологии.-157с.
5. Интернет обучение [Электронный ресурс] / Котляр Д.С. Использование CASE-средства ERwin для автоматизации проектирования и разработки базы данных – Режим доступа: http://royallib.com/read/ bezopasnost – Дата доступа: 15.12.2015.
6. Интернет обучение [Электронный ресурс] / Информационные системы и технологии – Режим доступа: <http://www.narfu.ru> – Дата доступа: 19.12.2015.
7. Интернет обучение [Электронный ресурс] / Нормализация структурны данных – Режим доступа: http://infostart.ru/public/269803/ – Дата доступа: 15.01.2016.
8. Интернет обучение [Электронный ресурс] / Программа компьютерного моделирования BpWin – Режим доступа: http://bourabai.kz/cm/bpwin.htm– Дата доступа: 11.11.2015.

# Генерация кода

Генерация кода происходит на основе диаграммы компонентов. Спроектированы все необходимые данные, классы и диаграммы, осуществляться генерация кода будет на языке java.

Были сгенерированы компоненты реализации таких классов как:

1. «Department»
2. «Order»
3. «Employee»
4. «Goods»
5. «Loader»

Содержимое сгенерированного кода в файле Department.java:

//Source file: D:\\1\\Department.java

public class Department

{

private int id;

private String Phone;

private String Name;

private int Order\_id;

private String Address;

/\*\*

\* @roseuid 58815236032E

\*/

public Department()

{

}

/\*\*

\* @return String

\* @roseuid 587B4EE703B3

\*/

public Department()

{

}

/\*\*

\* @return String

\* @roseuid 587B6AB402A5

\*/

public String print()

{

return null;

}

/\*\*

\* @return String

\* @roseuid 58814C070264

\*/

public String Report()

{

return null;

}

}

Содержимое сгенерированного кода в файле Order.java:

//Source file: D:\\1\\Order.java

public class Order

{

private int id;

private String Date;

private int Employee\_id;

private int Department\_id;

private String Report;

private int Goods\_id;

private int Price;

private int Summ;

/\*\*

\* @roseuid 5881520D0283

\*/

public Order()

{

}

/\*\*

\* @return String

\* @roseuid 587B51F80254

\*/

public Order()

{

}

/\*\*

\* @return String

\* @roseuid 587B51FD0344

\*/

public String print()

{

return null;

}

/\*\*

\* @return Boolean

\* @roseuid 587B5205038E

\*/

public Boolean close()

{

return null;

}

}

Содержимое сгенерированного кода в файле Employee.java:

//Source file: D:\\1\\Employee.java

public class Employee

{

private int id;

private String Phone;

private String Name;

private int Order\_id;

private String Report;

/\*\*

\* @roseuid 5881520D0207

\*/

public Employee()

{

}

/\*\*

\* @return String

\* @roseuid 587B50BC01A2

\*/

public Employee()

{

}

}

Содержимое сгенерированного кода в файле Goods.java:

//Source file: D:\\1\\Goods.java

public class Goods

{

private int id;

private int Summ;

private String Name;

private int Price;

private int Loader\_id;

/\*\*

\* @roseuid 5881520D0234

\*/

public Goods()

{

}

/\*\*

\* @return String

\* @roseuid 587B6AB403B3

\*/

public Goods()

{

}

}

Содержимое сгенерированного кода в файле Loader.java:

//Source file: D:\\1\\Loader.java

public class Loader

{

private int id;

private String Name;

private String Phone;

private String Department;

/\*\*

\* @roseuid 5881520D025B

\*/

public Loader()

{

}

/\*\*

\* @return String

\* @roseuid 588149120014

\*/

public Loader()

{

}

}