Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Специальность: «Программное обеспечение информационных технологий»

**КОНТРОЛЬНАЯ РАБОТА**

По курсу: «Технология разработки программного обеспечения»

На тему: «Моделирование интернет магазина»

Студент-заочник\_2\_ курса

Группы №\_\_581072 е

ФИО \_Савончик т

Александр Сергеевич т

Проверила: Бакунова О.М.

Минск, 2017

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ ………………………………………………………………………….3

1 ОПИСАНИЕ И ОБОСНОВАНИЕ ВЫБРАННОГО ЯЗЫКА ПРОГРАММИРОВАНИЯ………………………………………………………..….4

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА…… ….7

3 ЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ………………………………………….....9

3.1 Диаграмма вариантов использования……………………………………… …..9

3.2 Диаграмма классов………………………………………………………….…..10

3.3 Диаграмма деятельности……………………………………………………….11

3.4 Диаграмма коопераций………………………………………………………....12

3.5 Диаграмма последовательности ……………………………………………….13

3.6 Диаграмма развёртывания……………………………………………… ……..14

3.7 Диаграмма компонентов…………………………………………………….…15

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ…………………………… …16

Генерация кода………………………………………………………………….…..17

# ВВЕДЕНИЕ

Настоящее техническое задание распространяется на разработку и испытание автоматизированной системы «Интернет магазина». Предполагается, что использовать данную систему будут покупатели имеющие доступ в интернет.

Поход по магазинам может отнимать большое количество времени и дополнительных затрат на дорогу, при этом нет гарантий, что приехав в магазин покупатель сразу найдет подходящий товар.

Автоматизированная система обслуживанию клиентов позволит сэкономить время на поиски нужного товара, а также оставить или прочитать отзывы на товары и самое главное не выходя из дома оформить доставку до подъезда.

Этап проектирования структуры программы заключается в разработке детальной схемы будущей программы, на которой указываются классы, их свойства и методы, а также различные взаимосвязи между ними. Результатом данного этапа должна стать детализированная схема программы, на которой указываются все классы и взаимосвязи между ними в процессе функционирования программы. Согласно методологии объектно-ориентированного анализа и проектирования (ООАП), именно данная схема должна служить исходной информацией для написания программного кода.

Методология ООАП тесно связана с концепцией автоматизированной разработки программного обеспечения (Computer Aided Software Engineering, CASE).

Объектно-ориентированная методология (ООМ) создания автоматизированных систем состоит из следующих частей:

* объектно-ориентированный анализ (OOA),
* объектно-ориентированное проектирование (OOD),
* объектно-ориентированное программирование (OOР).

ООА - методология анализа сущностей реального мира на основе понятий класса и объекта, составляющих словарь предметной области, для понимания и объяснения того, как они (сущности) взаимодействуют между собой.

OOР - совокупность идей и понятий, определяющая стиль написания программ, в которой основными концепциями являются понятия объектов и классов.

ООD - методология проектирования, соединяющая в себе процесс объектной декомпозиции, опирающийся на выделение классов и объектов, и приемы представления моделей, отражающих логическую (структура классов и объектов) и физическую (архитектура моделей и процессов) структуру системы.

# 1 ОПИСАНИЕ И ОБОСНОВАНИЕ ВЫБРАННОГО ЯЗЫКА ПРОГРАММИРОВАНИЯ

На сегодняшний момент язык Java является один из самых распространенных и популярных языков программирования.

Ключевой особенностью языка Java является то, что его код сначала транслируется в специальный байт-код, независимый от платформы. А затем этот байт-код выполняется виртуальной машиной JVM (Java Virtual Machine). В этом плане Java отличается от стандартных интерпретируемых языков как PHP или Perl, код которых сразу же выполняется интерпретатором. В то же время Java не является и чисто компилируемым языком, как С или С++. Подобная архитектура обеспечивает кроссплатформенность и аппаратную переносимость программ на Java, благодаря чему подобные программы без перекомпиляции могут выполняться на различных платформах - Windows, Linux, Solaris и т.д. Для каждой из платформ может быть своя реализация виртуальной машины JVM, но каждая из них может выполнять один и тот же код. Java является языком с Си-подобным синтаксисом и близок в этом отношении к C/C++ и C#. Поэтому, если вы знакомы с одним из этих языков, то овладеть Java будет легче. Еще одной ключевой особенностью Java является то, что она поддерживает автоматическую сборку мусора. А это значит, что вам не надо освобождать вручную память от ранее использовавшихся объектов, как в С++, так как сборщик мусора это сделает автоматически за вас. Java является объектно-ориентированным языком. Он поддерживает полиморфизм, наследование, статическую типизацию. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений.

Основные ключевые моменты объединившиеся 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.

Программы, написанные на Java, имеют репутацию более медленных и занимающих больше оперативной памяти, чем написанные на языке C[[12]](https://ru.wikipedia.org/wiki/Java#cite_note-slow_java-12). Тем не менее, скорость выполнения программ, написанных на языке Java, была существенно улучшена с выпуском в 1997—1998 годах так называемого JIT-компилятора в версии 1.1 в дополнение к другим особенностям языка для поддержки лучшего анализа кода (такие, как внутренние классы, класс StringBuffer, упрощенные логические вычисления и т. д.). Кроме того, была произведена оптимизация виртуальной машины Java — с 2000 года для этого используется виртуальная машина [HotSpot](https://ru.wikipedia.org/wiki/HotSpot" \o "HotSpot). По состоянию на февраль 2012 года, код Java 7 приблизительно в 1.8 раза медленнее кода, написанного на языке Си.

Некоторые платформы предлагают аппаратную поддержку выполнения для Java. К примеру, микроконтроллеры, выполняющие код Java на аппаратном обеспечении вместо программной JVM, а также основанные на ARM процессоры, которые поддерживают выполнение байткода Java через опцию Jazelle.

Основные возможности языка Java:

* автоматическое управление памятью;
* расширенные возможности обработки исключительных ситуаций;
* богатый набор средств фильтрации ввода-вывода;
* набор стандартных коллекций: [массив](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2), [список](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), [стек](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B5%D0%BA) и т. п.;
* наличие простых средств создания сетевых приложений (в том числе с использованием [протокола](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB) [RMI](https://ru.wikipedia.org/wiki/RMI));
* наличие классов, позволяющих выполнять [HTTP](https://ru.wikipedia.org/wiki/HTTP)-запросы и обрабатывать ответы;
* встроенные в язык средства создания многопоточных приложений, которые потом были портированы на многие языки (например, python);
* унифицированный доступ к [базам данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85):

# 2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

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

IDEF0 — методология функционального моделирования (англ. function modeling) и графическая нотация, предназначенная для формализации и описания бизнес-процессов. Отличительной особенностью IDEF0 является её акцент на соподчинённость объектов.

Стандарт IDEF0 представляет организацию как набор модулей, здесь существует правило — наиболее важная функция находится в верхнем левом углу, кроме того есть правило стороны:

* стрелка входа приходит всегда в левую кромку активности,
* стрелка управления — в верхнюю кромку,
* стрелка механизма — нижняя кромка,
* стрелка выхода — правая кромка.

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).

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).

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.

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.

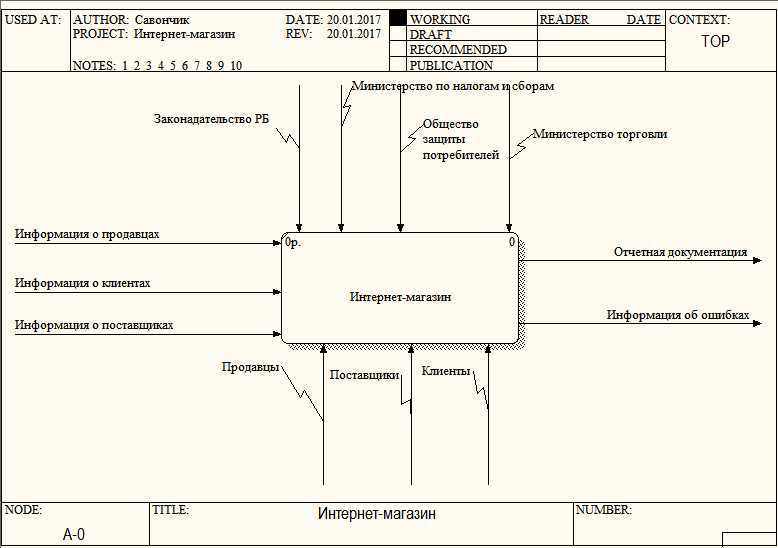


Рисунок 2.1- Контекстная диаграмма модели

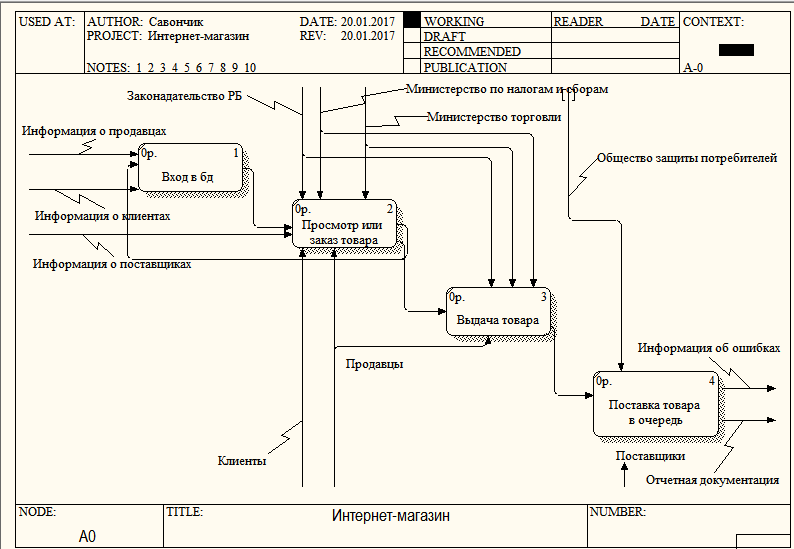


Рисунок 2.2 – Декомпозиция контекстной диаграммы функциональной модели

# 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

Диаграмма вариантов использования описывает функциональное назначение системы или, другими словами, то, что система будет делать в процессе своего функционирования. Она является исходным концептуальным представлением или концептуальной моделью системы в процессе ее проектирования и разработки.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества так называемых вариантов использования, предоставляемых системой множеству актеров или сущностей, взаимодействующих с системой. При этом актером (actor) или действующим лицом называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик. В свою очередь, вариант использования (use case) служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером.

Варианты использования определяют функциональные возможности. Каждый из них представляет определенный способ использования. Таким образом, каждый вариант использования соответствует последовательности действий для того, чтобы клиент мог получить определенный результат.

На рисунке представленном ниже, изображена диаграмма вариантов использования для интернет-магазина. Клиент - все люди, желающие воспользоваться услугами интернет-магазина; интернет-магазин – предоставляет онлайн услуги по продаже товара. Работники склада – проверяют наличие товара на складе и передают заказ курьеру. Курьер – доставляет заказ клиенту. Электронная система – проводит платежи.

Клиент обращается в интернет-магазин, для предоставления ему услуг, таких как заказ товара. Перед заказом товара, клиент проходит процедуру идентификации, и если нужно регистрируется в базе данных клиентов.

Основным вариантом использования служит “заказ товара”. Для получения товара, клиент смотрит в каталог товаров и выбирает нужный ему товар, поэтому “заказ товара”, включает (include) “просмотр каталога”. После просмотра каталога, клиенту необходимо сформировать заказ. Затем администратор проверяет наличие товара с помощью работников склада, и в том случае, если товар есть на складе, обновляет статус заказа и отправляет клиенту счёт. Чтобы оплатить счёт, клиент пользуется электронной системой, чтобы сделать это онлайн.

Тем временем работник склада передаёт заказ курьеру, а курьер доставляет товар клиенту

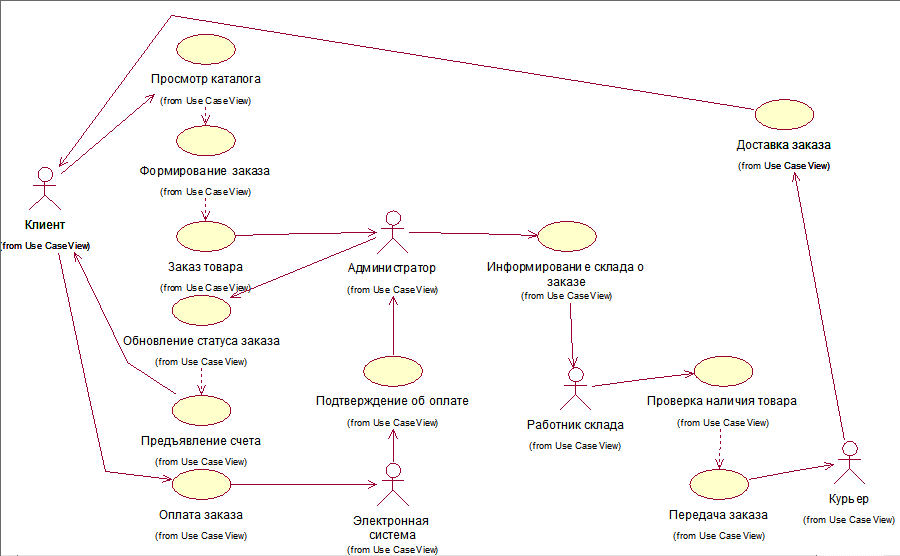


Рисунок 3.1 – Диаграмма вариантов использования

# 3.2 Диаграмма классов

Диаграмма классов позволяет создавать логическое представление системы, на основе которого создается исходный код описанных классов.

Этот тип диаграмм может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о временных аспектах функционирования системы. С этой точки зрения диаграмма классов является дальнейшим развитием концептуальной модели проектируемой системы.

Когда говорят о данной диаграмме, имеют в виду статическую структурную модель проектируемой системы. Поэтому диаграмму классов принято считать графическим представленном таких структурных взаимосвязей логической модели системы, которые не зависят от времени.

Диаграмма классов состоит из множества элементов, которые в совокупности отражают декларативные знания о предметной области. Эти знания интерпретируются в базовых понятиях языка UML, таких как классы, интерфейсы и отношения между ними и их составляющими компонентами.

Данная диаграмма показывает взаимосвязи между сущностями интерне-магазина, описывает внутреннюю структуру и типы отношений.

Работники – включает в себя всех работников магазина: администратор, электронная система, работники склада, курьер. Главным атрибутом класса является должность.

Администратор (Case worker) – является ключевой фигурой, так как взаимодействует с актерами в бизнес системе. Главным атрибутом является ФИО.

Электронная система – принимает платежи клиентов и информирует администратора о результате.

Работники склада – проверяют наличие товара и передают заказ в службу доставки. Главным атрибутом является: ФИО.

Курьер – принимает заказ от работников склада и доставляет заказ клиенту. Главным атрибутом является: ФИО.

Каталог (Business Entity) – перечень всего товара, представленного в интернет-магазине, и его цена. Главным атрибутом класса является: наименование товара.

Заказ – в заказе указывается только номер заказа, дата заказа и номер клиента. Главный атрибут: номер заказа.

Состав заказа– номер заказа, код товара и его количество. Главный атрибут: код товара.

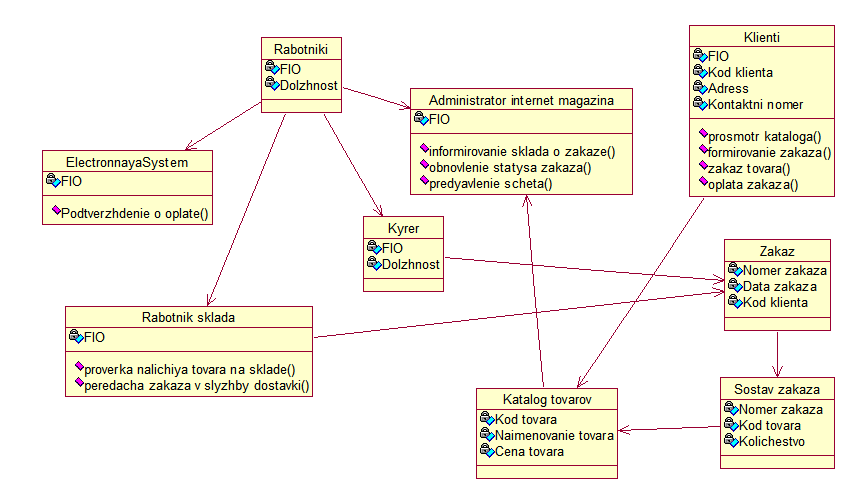
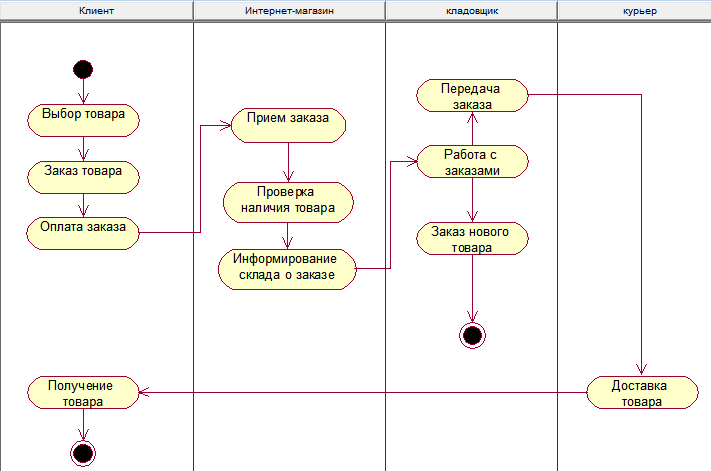


Рисунок 3.2 – Диаграмма классов

# 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.

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.

В отличие от диаграммы последовательности, на диаграмме кооперации изображаются только отношения между объектами, играющими определенные роли во взаимодействии. С другой стороны, на этой диаграмме не указывается время в виде отдельного измерения. Поэтому последовательность взаимодействий и параллельных потоков может быть определена с помощью порядковых номеров.

Поведение системы может описываться на уровне отдельных объектов, которые обмениваются между собой сообщениями, чтобы достичь нужной цели или реализовать некоторый сервис. С точки зрения аналитика или конструктора важно представить в проекте системы структурные связи отдельных объектов между собой. Такое статическое представление структуры системы как совокупности взаимодействующих объектов и обеспечивает диаграмма кооперации.

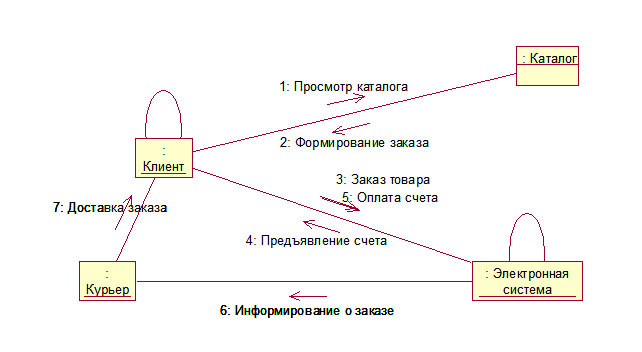


Рисунок 3.4 – Диаграмма коопераций

# 3.5 Диаграмма последовательности

Для моделирования взаимодействия объектов в языке UML используются соответствующие диаграммы взаимодействия. Одним из аспектов взаимодействия является время. Для представления временных особенностей передачи и приема сообщений между объектами используется диаграмма последовательности.

Хотя рассмотренные ранее диаграммы, такие как диаграмма вариантов использования и диаграмма классов, и используются для спецификации динамики поведения систем, время в явном виде в них не присутствует. Однако временной аспект поведения может иметь существенное значение при моделировании синхронных процессов, описывающих взаимодействия объектов. Именно для этой цели в языке UML используются диаграммы последовательности.

На рисунке 3.5 представлена диаграмма последовательности. Клиент зашёл на сайт интернет-магазина. Просмотрев каталог, он формирует заказ. Далее он заказывает товар, и электронная система предъявляет ему счёт. Клиент оплачивает счёт с помощью электронных денег. Через электронную систему курьер узнаёт о заказе и доставляет товар клиенту.

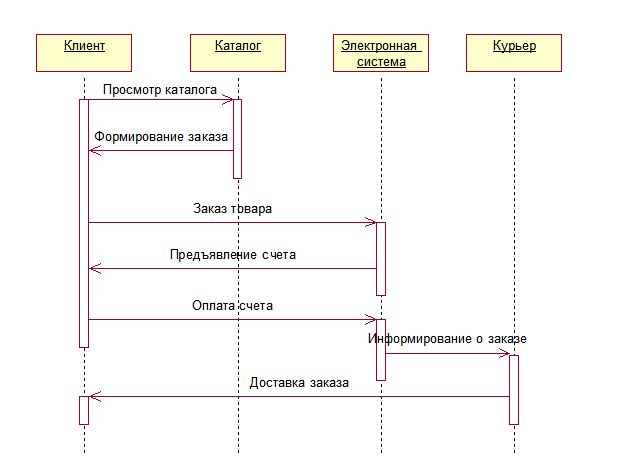


Рисунок 3.5 – Диаграмма последовательности

# 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).

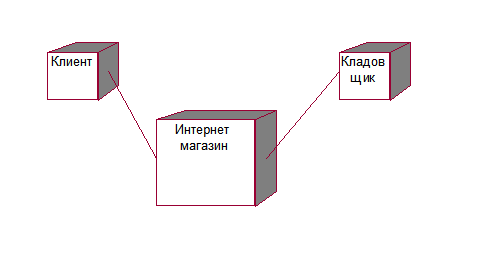


Рисунок 3.6 – Диаграмма развёртывания

# 3.7 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.

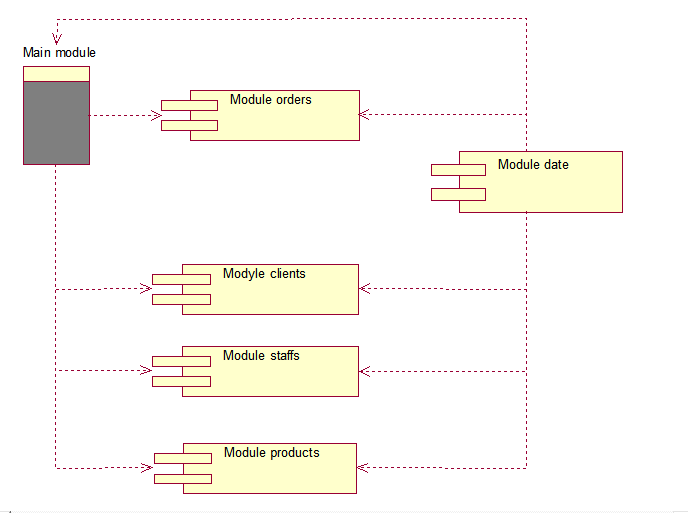


Рисунок 3.8 – Диаграмма компонентов

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Скопцов А. П. Локальная сеть в школе и малом офисе: создание и поддержка своими силами. – М.: [КУДИЦ-Пресс](http://www.knigalub.ru/publisher/141189.html), 2000. – 240 с.
2. Гук М. Аппаратные средства локальных сетей. – Спб.: Питер, 2000. – 544 c.
3. Капустин Н.М., Кузнецов П.М., Схиртладзе А.Г. Автоматизация производственных процессов в машиностроении. Учебник для втузов. – М.: Высшая школа, 2004. – 415 c.
4. [Минаев](http://www.knigalub.ru/author/127979.html) И. Я. 100% самоучитель. Локальная сеть своими руками. – М.: [Технолоджи-3000](http://www.knigalub.ru/publisher/136066.html), 2003. – 320 с.
5. Vito J. Amato Cisco Networking Essentials, Volume 2. – М.: Вильямс, 2002. – 464 с.
6. Единое окно образовательный ресурс [Электронный pecypc] Режим доступа:

http: // window.edu.ru., свободный .

1. Вишневский В.М. Теоретические основы проектирования компьютерных сетей. – М.: Техносфера, 2003. – 324 с.
2. Столингс В. Современные компьютерные сети. 2-е изд. – М.: Высшая школа, 2005. – 784 с.
3. Книга: Энциклопедия. – М.: Большая российская энциклопедия, 1999. – 800 с.
4. Ермишин П. Г. Основы экономической теории. – Спб.: Питер, 2004. – 340 с.

11 Леоненков. «Самоучитель UML».

12 Б. Хичков SYBASE: Настольная книга администратора. Лори, 2000 г. 448 с.

13 Р. Мюллер. [Базы данных и UML: Проектирование](javascript:parent.descr(window,%22d160587%22)).– Лори, 2002г. 432 с.

14 Фельдман С.К. Система программирования Java без секретов: Как создать безопасное приложение с "нуля". – Новый издательский дом" , 2005 г. , 347 с.

15 Дейтел П.Дж., Дейтел Х.М. Как програмировать на Java. Книга 2. Файлы, сети, базы данных. – "Бином" · 2005 г., 672 с.

16 http://www.avacco.ru/page.asp?code=electronniy\_arhiv

17 <http://www.java.alfamoon.com/>

18 <http://www.cfin.ru/vernikov/idef/idef0.shtml>

19 http://www.cfin.ru/vernikov/idef/idef1x.shtml

# Генерация кода

Содержимое сгенерированного файла JP.java

Private class ElectronnayaSystem

{

private int FIO;

/\*\*

@roseuid 5881375F02E8

\*/

public ElectronnayaSystem()

{

}

/\*\*

@roseuid 5880FB8301FC

\*/

public void PodtverzhdenieOOplate()

{

}

}

private class Klienti

{

private int FIO;

private int KodKlienta;

private int Adress;

private int KontaktniNomer;

public KatalogTovarov theKatalogTovarov;

/\*\*

@roseuid 5881375F036D

\*/

public Klienti()

{

}

/\*\*

@roseuid 5880FE4B029A

\*/

public void prosmotrKataloga()

{

}

/\*\*

@roseuid 5880FE5603BC

\*/

public void formirovanieZakaza()

{

}

/\*\*

@roseuid 5880FE610331

\*/

public void zakazTovara()

{

}

/\*\*

@roseuid 5880FE6E0146

\*/

public void oplataZakaza()

{

}

}

private class AdministratorInternetMagazina extends ElectronnayaSystem

{

private int FIO;

/\*\*

@roseuid 588137600065

\*/

public AdministratorInternetMagazina()

{

}

/\*\*

@roseuid 5880FD690269

\*/

public void informirovanieSkladaOZakaze()

{

}

/\*\*

@roseuid 5880FD7D0295

\*/

public void obnovlenieStatysaZakaza()

{

}

/\*\*

@roseuid 5880FD8B0298

\*/

public void predyavlenieScheta()

{

}

}

private class Kyrer

{

private int FIO;

private int Dolzhnost;

public Zakaz theZakaz;

/\*\*

@roseuid 588137600114

\*/

public Kyrer()

{

}

}

public class KatalogTovarov

{

private int KodTovara;

private int NaimenovanieTovara;

private int CenaTovara;

public AdministratorInternetMagazina theAdministratorInternetMagazina;

/\*\*

@roseuid 5881376001A8

\*/

public KatalogTovarov()

{

}

}

private class SostavZakaza

{

private int NomerZakaza;

private int KodTovara;

private int Kolichestvo;

public KatalogTovarov theKatalogTovarov;

/\*\*

@roseuid 58813760024B

\*/

public SostavZakaza()

{

}

}

private class Zakaz

{

private int NomerZakaza;

private int DataZakaza;

private int KodKlienta;

public SostavZakaza theSostavZakaza;

/\*\*

@roseuid 5881376002DA

\*/

public Zakaz()

{

}

}