КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ
И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
Кафедра информационных систем

О.А. НЕВЗОРОВА, Ч.Б.МИННЕГАЛИЕВА

ОСНОВЫ UML

Казань – 2021

УДК 004.4'2

Принято на заседании учебно-методической комиссии ИВМиИТ
Протокол от 18 марта 2021 года

Рецензенты:
кандидат физико-математических наук,
доцент кафедры информационных систем КФУ **Л.Э.Хайруллина**
старший преподаватель кафедры информационных систем КФУ
М.М.Аюпов

Невзорова О.А., Миннегалиева Ч.Б.
Основы UML / О.А. Невзорова, Ч.Б. Миннегалиева. – Казань: Казан. ун-т, 2021. – 43 с.

Учебно-методическое пособие знакомит с основами языка UML и особенностями процесса объектно-ориентированного анализа и проектирования информационных систем и их модулей. Рассматриваются типы диаграмм языка UML и практические рекомендации по их построению.

ОГЛАВЛЕНИЕ

1. Анализ требований к информационным системам 4
2. Основы UML 5
2.1. Диаграмма прецедентов 7
2.2. Диаграмма классов 11
2.3. Диаграмма деятельности 17
2.4. Диаграмма последовательностей 20
2.5. Диаграмма состояний 22
3. Примеры построения диаграмм 26
3.1. Пример построения диаграммы прецедентов 26
3.2. Пример построения диаграммы классов 28
3.3. Пример построения диаграммы деятельности 30
3.4. Пример построения диаграммы последовательностей 31
3.5. Пример построения диаграммы состояний 33
4. Задания для самостоятельной работы 34
Библиографический список 42

1. АНАЛИЗ ТРЕБОВАНИЙ К ИНФОРМАЦИОННЫМ СИСТЕМАМ

История развития систем, предназначенных для хранения и обработки информации с использованием ЭВМ, насчитывает уже более полувека. Информационная система в широком смысле – взаимосвязанная совокупность средств, методов и персонала, используемых для хранения, обработки и выдачи информации в интересах достижения поставленной цели. Основу информационной системы составляют база данных, программные средства, обеспечивающие логику обработки данных, интерфейс пользователя [подробнее см. 1, 3, 5, 8, 9].

Требования – это исходные данные, на основании которых проектируются и создаются автоматизированные информационные системы. Первичные данные поступают из различных источников, характеризуются противоречивостью, неполнотой, нечеткостью, изменчивостью. Требования нужны в частности для того, чтобы разработчик мог определить и согласовать с заказчиком временные и финансовые перспективы проекта автоматизации. Поэтому значительная часть требований должна быть собрана и обработана на ранних этапах создания информационной системы. На практике процесс сбора, анализа и обработки растянут во времени на протяжении всего жизненного цикла информационной системы [см. 9, 10, 14].

В своей основе требования – это то, что формулирует заказчик. Цель, которую он преследует – получить хороший конечный продукт: функциональный и удобный в использовании. Требования к продукту являются основополагающим классом требований.

Внедрение информационной системы на предприятии, в организации всегда преследует конкретные цели – такие, как сокращение сроков обработки информации, экономия накладных расходов. Современные информационные системы – это крупные программные системы, содержащие в себе множество модулей, функциональных, интерфейсных элементов, отчетов и т.д.

Обычно выделяют три уровня требований.

На верхнем уровне представлены так называемые бизнес-требования. Пример бизнес-требования: система должна сократить срок оборачиваемости обрабатываемых в Интернет-магазине заказов в три раза. Следующий уровень – уровень требований пользователей (user requirements). Пример требования пользователя: система должна представлять диалоговые средства для ввода исчерпывающей информации о заказе, последующей фиксации информации в базе данных и маршрутизации информации о заказе к сотруднику, отвечающему за его планирование и исполнение. Третий уровень – функциональный (functional requirements). Пример функциональных требований (или просто функций) по работе с электронным заказом: заказ может быть создан, отредактирован, удален и перемещен.

Функциональные требования регламентируют функционирование или поведение системы (behavioral requirements). Они отвечают на вопрос «Что должна делать система в тех или иных ситуациях?». Функциональные требования определяют основной объем работ разработчика, и устанавливают цели, задачи и сервисы, предоставляемые системой заказчику; записываются, как правило, при посредстве предписывающих правил: «система должна позволять библиотекарю редактировать формуляры».

Нефункциональные требования, соответственно, регламентируют внутренние и внешние условия или атрибуты функционирования системы. К ним относятся внешние интерфейсы, атрибуты качества, ограничения.

2. ОСНОВЫ UML

UML представляет собой объектно-ориентированный язык моделирования, обладающий следующими основными характеристиками:

- является языком визуального моделирования, который обеспечивает разработку репрезентативных моделей для организации взаимодействия заказчика и разработчика информационной системы (ИС), различных групп разработчиков ИС;
- содержит механизмы расширения и специализации базовых концепций языка.

Конструктивное использование языка UML основывается на понимании общих принципов моделирования сложных систем и особенностей процесса объектно-ориентированного анализа и проектирования (ООАП). Выбор выразительных средств для построения моделей сложных систем основывается на нескольких принципах.

Первым является принцип абстрагирования, который предписывает включать в модель только те аспекты проектируемой системы, которые имеют непосредственное отношение к выполнению системой своих функций или своего целевого предназначения. При этом все второстепенные детали опускаются, чтобы чрезмерно не усложнять процесс анализа и исследования полученной модели.

Вторым принципом построения моделей сложных систем является принцип многомодельности. Это значит, что никакое единственное представление сложной системы не является достаточным для адекватного выражения всех ее особенностей.

Еще одним принципом является принцип иерархического построения моделей сложных систем. Этот принцип предписывает рассматривать процесс построения модели на разных уровнях абстрагирования или детализации в рамках фиксированных представлений.

Таким образом, процесс ООАП можно представить как поуровневый спуск от наиболее общих моделей и представлений концептуального уровня к более частным и детальным представлениям логического и физического уровня. При этом на каждом из этапов ООАП данные модели последовательно дополняются все большим количеством деталей, что позволяет им более адекватно отражать различные аспекты конкретной реализации сложной системы.

Каждая из диаграмм UML символизирует определенный взгляд на систему. Чтобы понять систему в целом, необходима разработка и интеграция нескольких видов диаграмм, представляющих разные взгляды.

Можно выделить три типа UML-моделей.

1. Модель состояний, которая представляет статический взгляд на систему – это модель требований к данным. Модель состояний представляет структуры данных и отношения на них. Основной метод визуализации модели состояний состоит в использовании диаграммы классов.

2. Модель поведения, которая представляет операционный взгляд на систему – это модель функциональных требований. Для визуализации модели поведения существует несколько способов – диаграммы прецедентов, диаграммы последовательностей, диаграммы кооперации и диаграммы деятельности (видов деятельности).

3. Модель изменения состояний, которая представляет динамический взгляд на систему – это модель эволюции объектов со временем. Модель изменения состояний представляет возможные изменения состояний объекта. Основной метод визуализации модели изменения состояний заключается в использовании диаграммы состояний.

Для физического представления моделей систем в языке UML используются так называемые диаграммы реализации (implementation diagrams), которые включают в себя две отдельные канонические диаграммы: диаграмму компонентов и диаграмму развертывания [подробнее см. 3, 6, 11, 13, 15, 16].

1.1. Диаграмма прецедентов (Use case diagram)

Поведение системы – это ее реакция в ответ на внешние события. В языке UML внешне наблюдаемое и допускающее тестирование поведение фиксируется в виде прецедентов. Прецедент выполняет функцию, которую может наблюдать внешний субъект и которая может быть впоследствии отдельно протестирована в процессе разработки [подробнее см. 2, 3, 4].

Субъект (actor, эктор) – это некто или нечто (человек, машина и т.д.), взаимодействующее с прецедентом. Субъект взаимодействует с прецедентом, ожидая получить некий полезный результат.

Диаграмма прецедентов – это наглядное представление субъектов и прецедентов вместе с любыми дополнительными определениями и спецификациями. Диаграмма прецедентов представляет собой не просто некую

схему, а является полностью документированной моделью предполагаемого поведения системы.

Субъекты и прецеденты определяются в результате анализа функциональных требований. Прецеденты удовлетворяют функциональные требования за счет предоставления субъекту полезного результата. Типичное графическое изображение субъекта приведено на рис. 1.1.

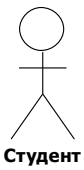

Рис.1.1. Субъект (actor, эктор)

Прецедент представляет собой некий целостный набор функций, имеющих определенную ценность для субъекта. Субъект, который не общается с прецедентом, не имеет смысла. Но может существовать прецедент, который не общается с субъектом. Например, прецеденты, которые обобщают или уточняют основной прецедент и не взаимодействуют непосредственно с субъектами. Они используются как внутренние в модели прецедентов и помогают основному прецеденту выработать результат, предоставляемый субъекту.

Чтобы вывести прецеденты, следует задаться вопросом: «Каковы обязанности субъекта по отношению к системе и чего он ожидает от системы?» Прецеденты также можно определить в результате непосредственного анализа функциональных требований. Во многих случаях функциональное требование отображается непосредственно в прецедент.

Графически прецедент изображается в виде эллипса, внутри или ниже которого помещается имя прецедента (рис.1.2).

Рис.1.2. Прецедент

На диаграммах прецедентов, кроме связей между действующими лицами и прецедентами, возможно использование еще двух видов связей между прецедентами: «использование» (<<include>>, включение) и «расширение» (<<extend>>) (рис. 1.3, 1.4). Связь типа «расширение» применяется, когда один прецедент подобен другому, но несет несколько большую функциональную нагрузку. Ее следует применять при описании изменений в нормальном поведении системы. Связь типа «использование» позволяет выделить некий фрагмент поведения системы и включать его в различные прецеденты без повторного описания.

Рис.1.3. Расширение прецедента

На рис.1.3 показано, что прецедент «Предоставить результаты (экзаменов)» может «расширить» прецедент «Предоставить инструкции (по записи)». Первый прецедент не всегда расширяет последний прецедент. Для новых студентов результаты экзаменов неизвестны. Поэтому отношение моделируется с использованием стереотипа расширения, а не включения.

На рис.1.4 отношение <<include>> было установлено от прецедента «ввести программу (обучения)» к прецеденту «Проверить программу (обучения)». Отношение <<include>> означает, что первый из прецедентов всегда включает последний. Как только программа введена, она проверяется на предмет конфликтов расписания и пр.

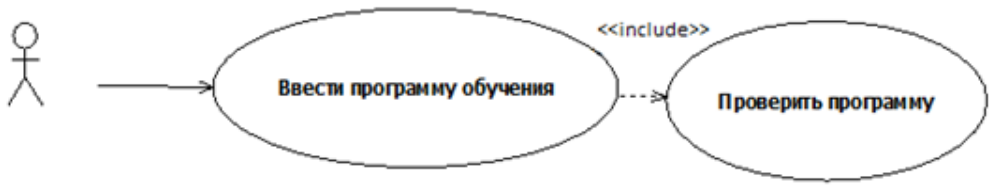

Рис.1.4. Включение

Главное назначение диаграммы прецедентов заключается в формализации функциональных требований к системе и возможности согласования полученной модели с заказчиком на ранней стадии проектирования.

С системно-аналитической точки зрения построение диаграммы прецедентов специфицирует не только функциональные требования к проектируемой системе, но и выполняет исходную структуризацию предметной области. Важно понимать, что все сервисы системы должны быть явно определены на диаграмме прецедентов, и никаких других сервисов, которые отсутствуют на данной диаграмме, проектируемая система не может выполнять по определению.

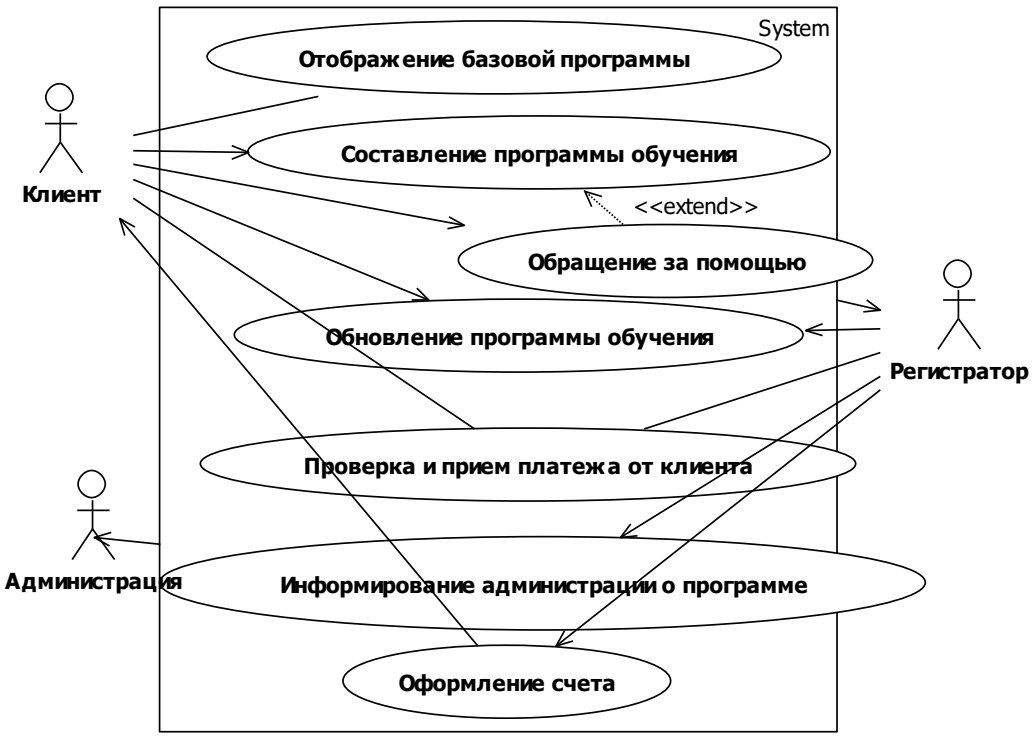

Рис.1.5. Диаграмма прецедентов для модуля «Запись на курсы»

1.2. Диаграмма классов (Class diagram)

Классы – это базовые элементы любой объектно-ориентированной системы. Классы представляют собой описание совокупностей однородных объектов с присущими им свойствами – атрибутами, операциями, отношениями и семантикой [см. 3, 5, 13].

В рамках модели каждому классу присваивается уникальное имя, отличающее его от других классов.

Атрибут – это свойство класса, которое может принимать множество значений. Атрибут имеет имя и отражает некоторое свойство моделируемой сущности, общее для всех объектов данного класса. Класс может иметь произвольное количество атрибутов.

Операция – реализация функции, которую можно запросить у любого объекта класса. Операция показывает, что можно сделать с объектом. Исполнение операции часто связано с обработкой и изменением значений атрибутов объекта, а также изменением состояния объекта. Объекты отвечают за собственную «судьбу» и обычно поддерживают четыре элементарных операций: «Создать» (Create), «Читать» (Read), «Обновить» (Update), «Удалить» (Delete). CRUD-операции позволяют другим объектам отправлять сообщение объекту с требованием выполнить следующие действия:

- Создать новый экземпляр объекта.
- Получить доступ к состоянию объекта.
- Модифицировать состояние объекта.
- Объекту уничтожить самого себя.

На рис. 1.6 приведено графическое изображение класса «Преподаватель» в нотации UML. Графически класс представляется в виде прямоугольника с тремя отделениями, разделенными горизонтальными линиями. Верхнее отделение хранит имя класса. Среднее отделение содержит объявления всех атрибутов класса. Нижнее отделение содержит определение операций.

-

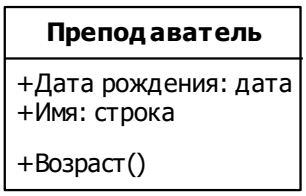

Рис. 1.6. Изображение класса в UML

Видимость свойства указывает на возможность его использования другими классами. Один класс может «видеть» другой, если тот находится в области действия первого и между ними существует явное или неявное отношение. В языке UML определены три уровня видимости:

public (общий) – любой внешний класс, который «видит» данный, может пользоваться его общими свойствами. Обозначаются знаком " + " перед именем атрибута или операции;

protected (защищенный) – только любой потомок данного класса может пользоваться его защищенными свойствами. Обозначаются знаком « # »;

private (закрытый) – только данный класс может пользоваться этими свойствами. Обозначаются символом « – ».

Принципы видимости операций не отличаются от принципов видимости атрибутов. Видимость операции определяет, является ли операция видимой объектам классов, отличных от класса, который определяет операцию. Если она видима, то ее видимость является открытой. В противном случае она будет закрытой. Как правило, большинство операций объектно-ориентированной системы обладают открытой видимостью. Чтобы объект мог предоставить сервис внешнему миру, операция «обслуживания» должна быть видима. Но большинство объектов имеют также несколько внутренних операций. Видимость этих операций должна быть закрыта. Они доступны только объектам класса, в котором определены.

Следует различать видимость операции и область действия операции. Операцию можно вызвать на объекте-экземпляре или же на объекте-классе. В первом случае говорят, что операция обладает областью действия экземпляра.

Во втором случае – областью действия класса. Например, операция, предназначенная для отыскания среднего балла студента, обладает областью действия экземпляра, а операция вычисления среднего балла студентов группы обладает областью действия класса.

Чтобы выявить классы, полезно задать вопросы:

- Является ли понятие «вместилищем данных»?
- Обладает ли отдельными атрибутами, способными принимать различные значения?
- Можно ли для него создать множество объектов-экземпляров?
- Входит ли определенный класс в границы системы?

Классы в UML изображаются на диаграммах классов, которые позволяют описать систему в статическом состоянии – определить типы объектов системы и различного рода статические связи между ними.

Классы отображают типы объектов системы. Между классами возможны различные отношения.

Ассоциации. Отношение ассоциации устанавливает связь между объектами данных классов. Объекты, которым требуется взаимодействовать друг с другом, могут использовать установленную связь. Порядок ассоциации определяет количество классов, соединенных с помощью ассоциации. Наиболее часто встречаются ассоциации второго порядка (бинарная ассоциация). Если ассоциацию определить на единственном классе, она называется унарной или сингулярной.

Кратность ассоциации определяет, сколько объектов могут занимать позицию, указанную ролевым именем. Кратность говорит о том, сколько объектов целевого класса (указываемых ролевым именем) может быть ассоциировано с одним объектом исходного класса.

Кратность обозначается в виде диапазона целых чисел $n_1..n_2$. Число n_1 определяет минимальное количество связываемых объектов, а n_2 – максимальное количество. Если максимальное количество неизвестно, то может быть поставлена звездочка *. Если точное минимальное количество заранее

неизвестно, но мы знаем, что в отношении может участвовать несколько объектов, этот параметр можно не указывать. Наиболее часто встречаются следующие значения кратности:

0 .. 1 0 .. * 1 .. 1 1 .. * *.

На рис.1.7 показаны две ассоциации на объектах классов «Преподаватель» и «Курс». Одна из ассоциаций фиксирует закрепление преподавателей за текущими учебными курсами. Другая определяет, кто из преподавателей отвечает за учебный курс. Преподаватель может вести несколько учебных курсов или не вести ни одного. Учебный курс ведут один или несколько преподавателей. Один из этих преподавателей отвечает за курс. В общем случае преподаватель может отвечать за несколько курсов или не отвечать вовсе. Один и только один преподаватель руководит учебным курсом.

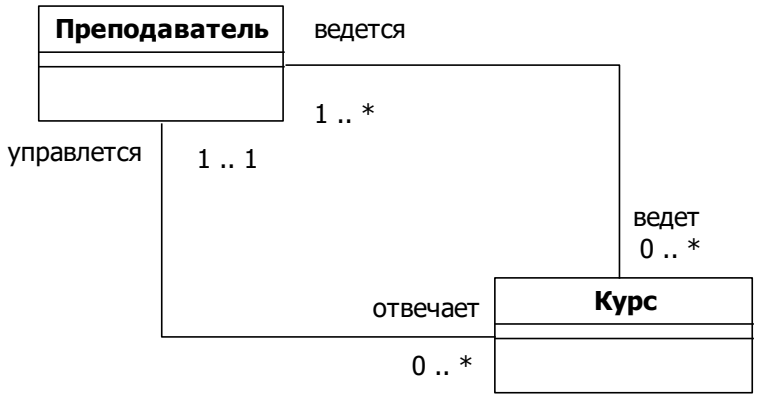

Рис. 1.7. Ассоциации

«Нулевая» минимальная кратность означает необязательную принадлежность объекта ассоциации. «Единичная» кратность означает обязательную принадлежность. Например, курс должен проводиться под руководством преподавателя.

Иногда ассоциация обладает своими собственными атрибутами. В качестве модели подобной ассоциации необходимо использовать класс, поскольку атрибуты могут быть определены только в классе. Каждый объект ассоциативного класса обладает значениями атрибутов и связями с ассоциированными классами.

На рис.1.8 показан ассоциативный класс «Оценка». Объект класса «Оценка» хранит общий балл и оценку, полученную студентом («Студент») по учебному курсу («Курс»).

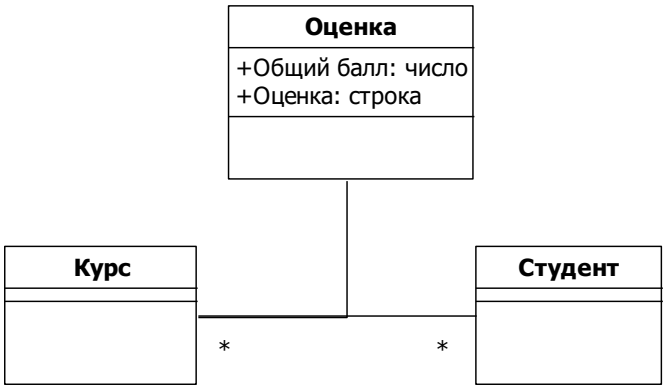

Рис.1.8. Ассоциативный класс

Агрегация и композиция. Агрегация – это отношение вида часть-целое между классом, который представляет собрание компонент (класс-супермножество) и классами, представляющими компоненты (класс-подмножество). Класс-супермножество содержит один или более классов-подмножеств. Свойство включения может быть сильным (агрегация по значению) или слабым (агрегация по ссылке). В UML агрегация по значению называется композицией. Композиция обладает дополнительным свойством зависимости по существованию. Объект класса-подмножества не может существовать в отсутствие связи с объектом класса-супермножества.

Композиция обозначается заполненным ромбовидным значком на конце ассоциативной связи. Агрегация, не являющаяся композицией, помечается незаполненным ромбом. Пустой ромб можно использовать, если в ходе моделирования вопрос о том, является ли агрегация композицией или нет, следует оставить открытым. На рис. 1.9 слева показана композиция, справа – обычная агрегация. Всякий объект «Книга» является композицией объектов «Глава». Объект «Глава» не обладает собственной независимой жизнью. Объект «Компьютер» может существовать вне объекта «Аудитория».

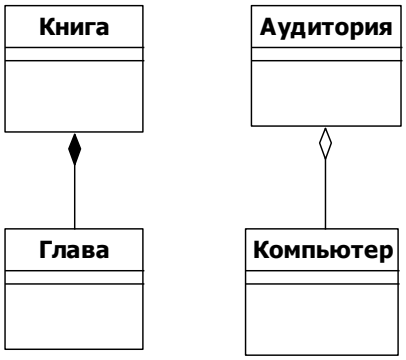

Рис.1.9. Композиция и агрегация

Обобщение. Обобщение – это отношение между более общим классом (суперкласс или родительский класс) и более специфическим видом класса (подкласс или дочерний класс). Обобщение делает невозможным переопределение уже заданных свойств. Атрибуты и операции, уже определенные для суперкласса, могут повторно использоваться в подклассе. Говорят, что подкласс наследует атрибуты и методы его родительского класса. Обобщение изображается в виде незаполненного треугольника на конце линии отношения, присоединенной к родительскому классу. На рис.1.10 «Сотрудник» является суперклассом, а «Преподаватель» – подклассом. Класс «Преподаватель» наследует все атрибуты и операции класса «Сотрудник». Наследуемые свойства могут явно не показываться в прямоугольнике, обозначающем подкласс. Наследование применимо к классам, а не объектам. Оно применимо по отношению к типам, а не значениям.

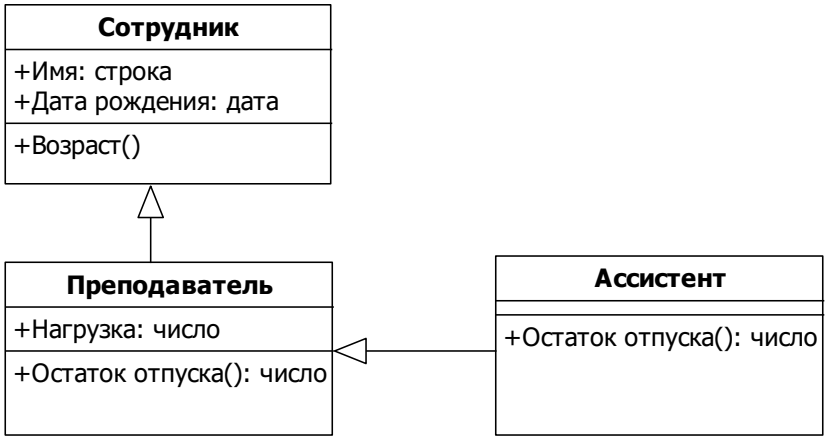

Рис.1.10. Обобщение

На рис.1.10 класс «Преподаватель» наследует определения атрибутов «Имя» и «Дата рождения». Объект класса «Преподаватель» может быть позже реализован с использованием значений этих атрибутов (поскольку атрибуты существуют наряду с атрибутом «Нагрузка»).

Метод, унаследованный подклассом, напрямую используется этим подклассом. Операция «Возраст» работает аналогично в объектах классов «Сотрудник» и «Преподаватель». Однако, иногда необходимо, чтобы операции в подклассе были замещены. Например, допустим, что ассистент имеет право на дополнительные дни отпуска. Тогда имя операции в подклассе дублируется. Теперь операция «Остаток отпуска» замещена («модифицирована»). Существует две реализации (два метода) операции. Если отправить сообщение объекту «Преподаватель» или объекту «Ассистент», то будут выполняться различные методы. Говорят, что такая операция полиморфна.

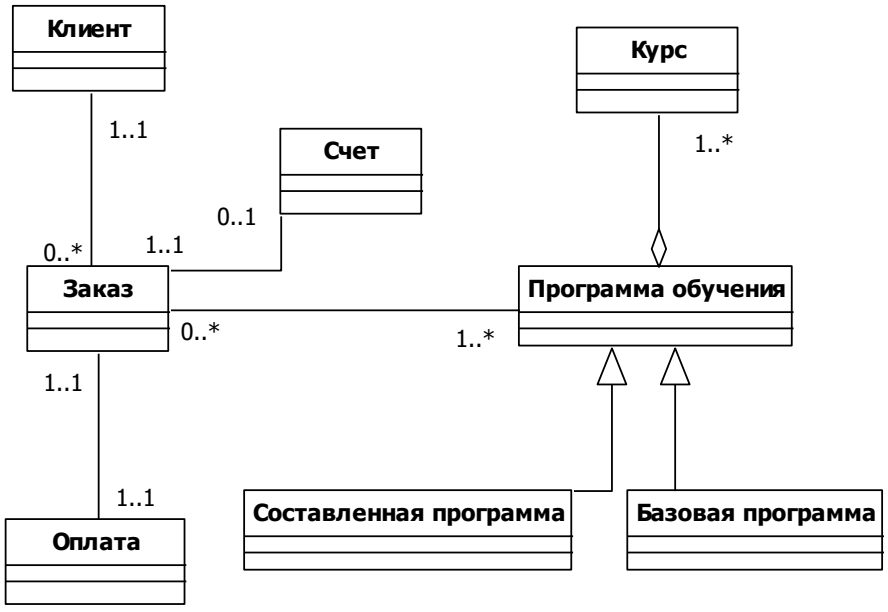

Рис.1.11. Пример диаграммы классов для модуля «Запись на курсы»

2.3. Диаграмма деятельности (Activity diagram)

Диаграмма деятельности (видов деятельности) – диаграмма, на которой показано разложение некоторой деятельности на её составные части. Под деятельностью понимается спецификация исполняемого поведения в виде

координированного последовательного и параллельного выполнения подчинённых элементов – вложенных видов деятельности и отдельных действий, соединённых между собой потоками, которые идут от выходов одного узла к входу другого. Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия, а дугами – переходы от одного состояния действия к другому [3, 4, 13].

Состояние действия является специальным случаем состояния с некоторым входным действием и, по крайней мере, одним выходящим из состояния переходом. Графически состояние действия изображается фигурой, напоминающей прямоугольник (рис.1.12). Внутри этой фигуры записывается выражение действия (action-expression), которое должно быть уникальным в пределах одной диаграммы деятельности.

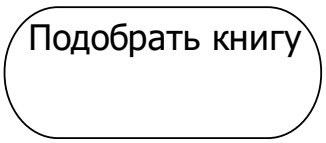

Рис.1.12. Состояние действия

Ветвление на диаграмме деятельности обозначается небольшим ромбом, внутри которого нет никакого текста. Начало обозначается полностью закрашенным кружочком. Для распараллеливания вычислений используется специальный символ для разделения и слияния параллельных вычислений или потоков управления (рис.1.13).

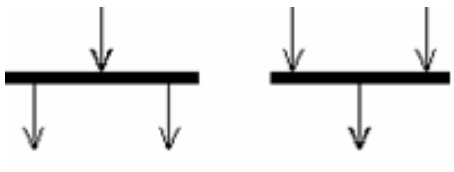

Рис.1.13. Раздел и слияние потоков

Для моделирования бизнес-процессов 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.1.14).

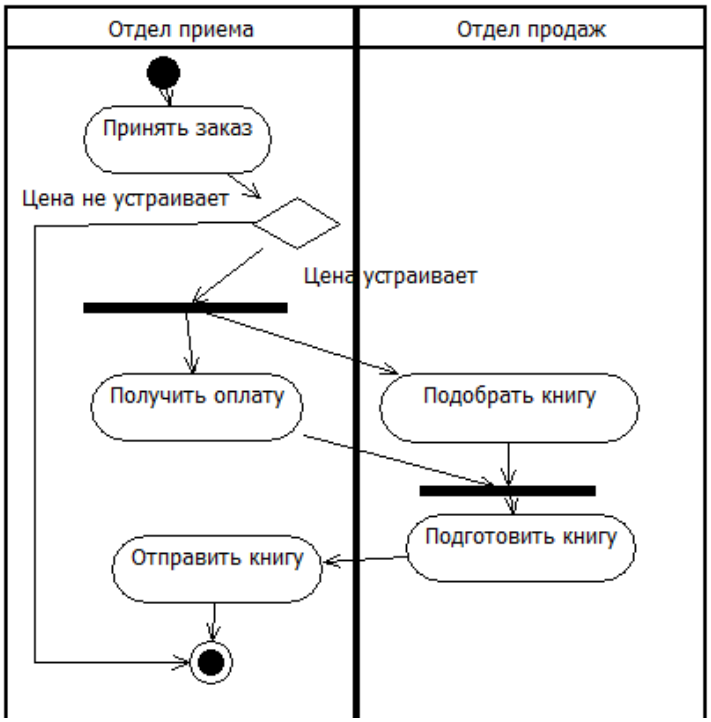

Рис.1.14. Пример диаграммы деятельности

Таким образом, диаграмма деятельности показывает шаги вычисления. Каждый шаг соответствует состоянию, в котором что-либо выполняется. Диаграмма описывает, какие шаги выполняются последовательно, а какие – параллельно.

Диаграммы деятельности играют важную роль в понимании процессов реализации алгоритмов выполнения операций классов и потоков управления в моделируемой системе. Используемые для этой цели блок-схемы алгоритмов обладают серьёзными ограничениями в представлении параллельных процессов и их синхронизации. Применение дорожек и объектов открывает позволяет более наглядно представить бизнес-процессы, позволяя специфицировать деятельность подразделений компаний и фирм.

2.4 Диаграмма последовательностей (Sequence diagram)

Этот вид диаграмм используется для точного определения логики сценария выполнения прецедента. Для каждого прецедента часто строится отдельная диаграмма последовательностей. Но диаграммы последовательностей могут отсутствовать для операций, которые охватывают более одного прецедента [подробнее см. 1, 3, 6, 13].

Взаимодействие (interaction) представляет собой набор сообщений, свойственных поведению некоторой системы, которыми обмениваются объекты в соответствии с установленными между ними связями. Диаграммы последовательностей отображают типы объектов, взаимодействующих при исполнении прецедентов, сообщения, которые они посылают друг другу, и любые возвращаемые значения, ассоциированные с этими сообщениями. Диаграмма последовательностей представляется двумерным графом. Объекты располагаются по горизонтали. Линии со стрелками и надписями названий методов означают вызов метода у объекта. Последовательности сообщений располагаются сверху вниз по вертикали. Каждая вертикальная линия называется линией жизни объекта. Исследование взаимодействий между классами может привести к выявлению операций. Каждое сообщение обращается к операции в вызываемом объекте.

Показывать на диаграмме возврат управления от объекта-получателя объекту-отправителю необязательно. Стрелка, указывающая на объект-получатель, предполагает автоматический возврат управления отправителю (рис.1.15).

Рис.1.15. Пример диаграммы последовательностей («Обновление программы обучения»)

Рассмотрим диаграмму, показанную на рис.1.15. Клиент решает обновить программу обучения в системе дистанционного обучения. Программа обучения составляется из отдельных курсов. Получив их, обновленная программа обучения отображается в диалоговом окне.

На диаграмме последовательностей изображаются только те объекты, которые непосредственно участвуют во взаимодействии и не показываются возможные статические ассоциации с другими объектами. Ключевым моментом является именно динамика взаимодействия объектов во времени.

Цель взаимодействия в контексте языка UML заключается в том, чтобы специфицировать коммуникацию между множеством взаимодействующих объектов. Каждое взаимодействие описывается совокупностью сообщений, которыми участвующие в нем объекты обмениваются между собой. В этом смысле сообщение представляет собой законченный фрагмент информации, который отправляется одним объектом другому. При этом прием сообщения инициирует выполнение определенных действий, направленных на решение отдельной задачи тем объектом, которому это сообщение отправлено.

Таким образом, сообщения не только передают некоторую информацию, но и требуют или предполагают от принимающего объекта выполнения ожидаемых действий. Сообщения могут инициировать выполнение операций объектом

соответствующего класса, а параметры этих операций передаются вместе с сообщением. На диаграмме последовательностей все сообщения упорядочены по времени своего возникновения в моделируемой системе.

Построение диаграммы последовательностей целесообразно начинать с выделения из всей совокупности тех классов, объекты которых участвуют в моделируемом взаимодействии. После этого все объекты наносятся на диаграмму с соблюдением некоторого порядка инициализации сообщений. Когда объекты визуализированы, можно приступать к спецификации сообщений. При этом следует учитывать те роли, которые играют сообщения в системе.

2.5. Диаграмма состояний

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 [3, 6, 7, 8, 13].

Главное предназначение этой диаграммы – описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла.

Диаграмма состояний является графом специального вида, который представляет некоторый автомат. 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.

Автомат в языке UML представляет собой некоторый формализм для моделирования поведения элементов модели и системы в целом. Основными

понятиями, входящими в формализм автомата, являются состояние и переход. Главное различие между ними заключается в том, что длительность нахождения системы в отдельном состоянии существенно превышает время, которое затрачивается на переход из одного состояния в другое. Предполагается, что в пределе время перехода из одного состояния в другое равно нулю. Другими словами, переход объекта из состояния в состояние происходит мгновенно.

В общем случае автомат представляет динамические аспекты моделируемой системы в виде ориентированного графа, вершины которого соответствуют состояниям, а дуги – переходам. При этом поведение моделируется как последовательное перемещение по графу состояний от вершины к вершине по связывающим их дугам с учетом их ориентации. Формализм обычного автомата основан на выполнении следующих обязательных условий:

1. Автомат не запоминает историю перемещения из состояния в состояние.

2. В каждый момент времени автомат может находиться в одном и только в одном из своих состояний.

3. Длительность нахождения автомата в том или ином состоянии, а также время достижения того или иного состояния не специфицируются.

4. Количество состояний автомата должно быть обязательно конечным (в языке UML рассматриваются только конечные автоматы).

5. Граф автомата не должен содержать изолированных состояний и переходов. Это условие означает, что для каждого из состояний, кроме начального, должно быть определено предшествующее состояние. Каждый переход должен обязательно соединять два состояния автомата. Допускается переход из состояния в себя, такой переход еще называют «петлей».

6. Автомат не должен содержать конфликтующих переходов, т. е. таких переходов из одного и того же состояния, когда объект одновременно может перейти в два и более последующих состояния (кроме случая параллельных подавтоматов).

В UML 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.

Следует заметить, что не каждый атрибут класса может характеризовать его состояние. Как правило, имеют значение только такие свойства элементов системы, которые отражают динамический или функциональный аспект ее поведения.

Поскольку состояние системы является составной частью процесса ее функционирования, рекомендуется использовать такие имена, как «печатает», «ожидает», «занят», «свободен», «передано» и т.д.

Простой переход 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.

Переход осуществляется при наступлении некоторого события: окончания выполнения деятельности, 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.

В языке UML 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, если оно есть, записывается в прямых скобках после события и представляет собой некоторое булевское выражение: «истина» или «ложь». 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.

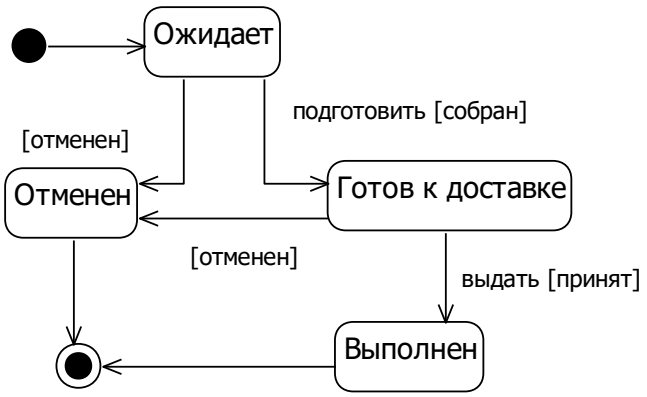

Рис. 1.16. Пример диаграммы состояний для класса «Заказ» (Интернет магазин по продаже книг)

3. ПРИМЕРЫ ПОСТРОЕНИЯ ДИАГРАММ

Краткое описание: Производитель предлагает возможность заказа проекционного оборудования через сайт. Клиент может выбрать стандартный комплект оборудования или собрать его из отдельных устройств. Для каждого комплекта система подсчитывает цену. Клиент заполняет информацию о себе (адрес, данные об оплате и т.д.). После этого система отправляет подтверждение на электронную почту. Также система предоставляет возможность получения консультации, проверяет наличие оборудования на складе, формирует необходимые документы. (Примечание: в этом примере мы не будем детально рассматривать процесс оплаты. Т.к. при подключении платежной системы переход финансов осуществляется с карты покупающего на счет при участии банковского посредника. Для платежей требуется интеграция специального сервиса). Рассмотрим построение диаграмм UML, отметим, что с примерами можно ознакомиться в [3, 7].

3.1. Пример построения диаграммы прецедентов

Субъекты и прецеденты определяются в результате анализа функциональных требований. Функциональные требования воплощаются в прецедентах. Прецедент – это внешне наблюдаемое и допускающее тестирование поведение. Субъект взаимодействует с прецедентом, ожидая получить некоторый полезный результат. В нашем примере субъекты, которые явно представлены в описании – это *Клиент, Продавец, Склад*. Чтобы определить прецеденты, зададимся вопросом: «Каковы обязанности субъекта по отношению к системе и что он ожидает от системы?». Во многих случаях функциональное требование отображается в прецедент (Таблица 1).

Таблица 1

№	Требование	Субъект	Прецедент
1	Клиент на сайте производителя выбирает стандартный комплект оборудования	Клиент	Отображение стандартного комплекта
2	Клиент выбирает устройства, составляющие подходящий ему комплект	Клиент	Составление комплекта оборудования

3	Клиент может заказать комплект или сначала обратиться за консультацией к продавцу.	Клиент Продавец	Заказ комплекта оборудования Обращение за консультацией к продавцу
4	Клиент заполняет данные о заказе	Клиент Продавец	Проверка деталей заказа
5	Продавец отправляет на склад требование, содержащее детали заказа	Продавец Склад	Информирование склада о заказе
6	Детали заказа отправляются клиенту, клиент может проверить состояние заказа на сайте	Продавец Клиент	Обновление состояния заказа
7	Склад получает документы от продавца и отправляет заказанный комплект клиенту	Продавец Склад	Печать документов

Диаграмма прецедентов приписывает прецеденты к субъектам. Эта диаграмма – основной метод визуализации для модели поведения системы.

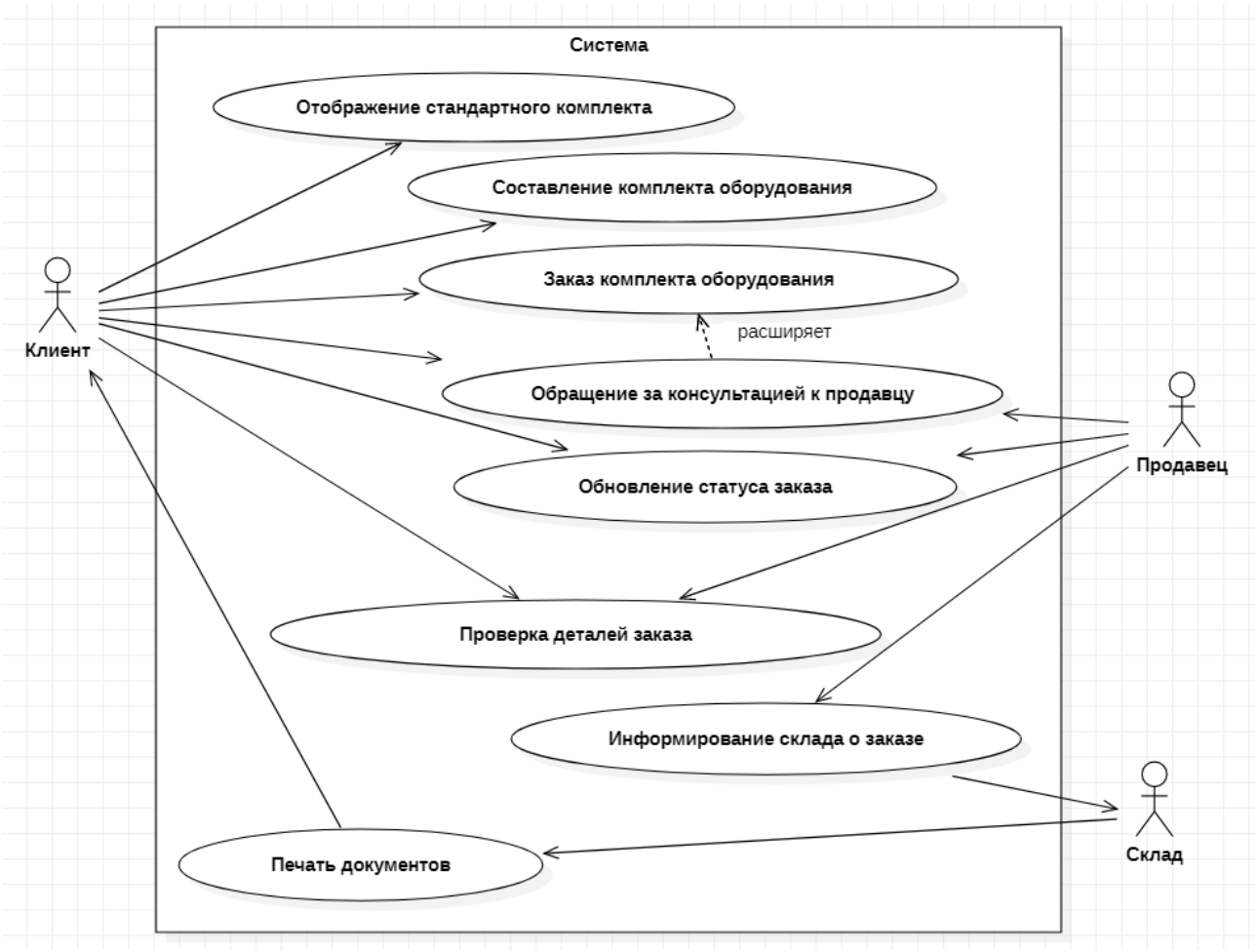

Рис. 2.1. Диаграмма прецедентов

3.2. Пример построения диаграммы классов

Класс – это дескриптор множества объектов, обладающих одинаковым набором атрибутов и операций. Он служит в качестве шаблона для создания объектов. Определение внутреннего состояния системы дается в модели классов. Классы-сущности определяют существо любой информационной системы, представляют постоянно хранимые объекты базы данных. В таблице 2 функциональным требованиям поставлены в соответствие классы-сущности.

Таблица 2

№	Требование	Класс-сущность
1	Клиент на сайте производителя выбирает стандартный комплект оборудования	Клиент Комплект оборудования Стандартный комплект
2	Клиент выбирает устройства, составляющие подходящий ему комплект	Клиент Собранный комплект Устройство
3	Клиент может заказать комплект или сначала обратиться за консультацией к продавцу.	Клиент Собранный комплект Заказ Продавец
4	Клиент заполняет данные о заказе	Клиент Заказ Поставка Счет Платеж
5	Продавец отправляет на склад требование, содержащее детали заказа	Клиент Заказ Продавец Собранный комплект Устройство
6	Детали заказа отправляются клиенту, клиент может проверить состояние заказа на сайте	Клиент Заказ Состояние заказа
7	Склад получает документы от продавца и отправляет заказ клиенту	Счет Отгрузка

Выделение классов представляет собой итеративную задачу, и первоначальный перечень обычно претерпевает изменения. Повторим вопросы, которые могут помочь при определении того, являются ли понятия, присутствующие в требованиях, искомыми классами:

1. Является ли понятие «вместилищем данных»?

2. Обладает ли оно отдельными атрибутами, способными принимать разные значения?

3. Можно ли для него создать множество объектов-экземпляров?

4. Входит ли оно в границы системы?

Чтобы уточнить первоначально определенный перечень в рассматриваемом случае, можно ответить на вопросы:

1. В чем различие между классами *Собранный комплект* и *Заказ*? Отметим, что не планируется хранение информации о собранном комплекте до тех пор, пока заказ не оформлен.

2. Необходим ли класс *Доставка*, если поставка комплекта входит в обязанности склада и выходит за рамки нашего приложения?

3. *Устройство* является отдельным классом или только атрибутом класса *Собранный комплект*?

4. Класс *Состояние заказа* необходим или же он будет являться атрибутом класса *Заказ*?

5. Является ли понятие *Продавец* самостоятельным классом или атрибутом классов *Заказ* и *Счет*?

После анализа оставим классы *Клиент, Заказ, Оплата, Счет, Устройство, Комплект*, запишем некоторые основные атрибуты этих классов (имя, адрес, цена и т.д.). Определим ассоциации между ними. Заказ поступает от одного клиента (1..1), но клиент может разместить несколько заказов (0..*). Заказ не принимается до тех пор, пока не определяются реквизиты платежа (1..1). Счет всегда связан с одним заказом (1..1). Заказ делается на один или несколько комплектов (1..*), комплект может не заказываться или заказываться несколько раз (0..*).

Комплект состоит из одного или нескольких устройств (1..*). В этом случае между *Комплект* и *Устройство* мы определяем более сильную форму ассоциативной связи – агрегацию (в конце линии незаполненный ромб). Агрегация – это отношение вида часть-целое между классом, который представляет собрание компонент и классом, представляющим компоненты.

Обобщение представляет собой отношение между более общим классом (*Комплект*) и более специфическими видами класса (*Собранный комплект и Стандартный комплект*), изображается в виде незаполненного треугольника на конце линии.

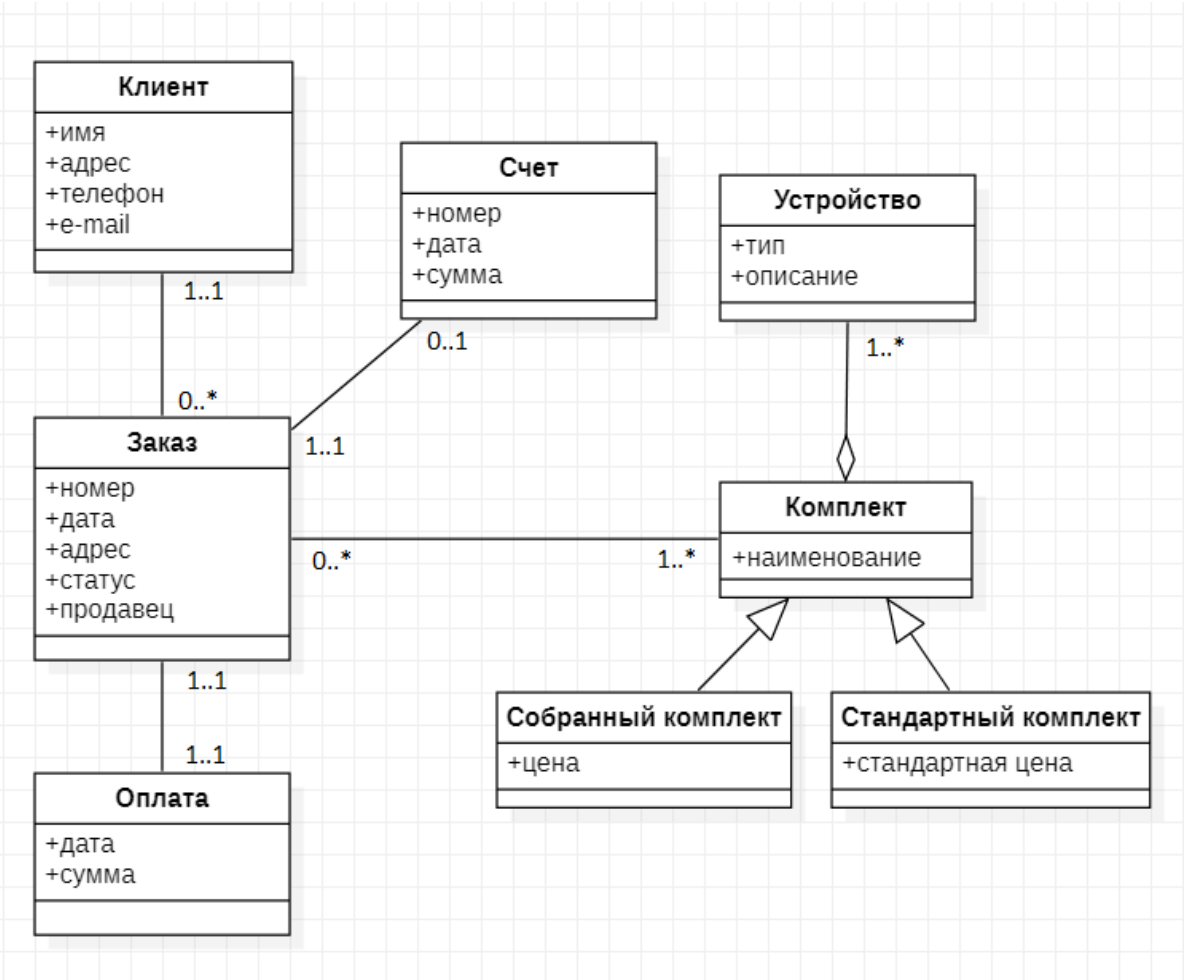

Рис. 2.2. Диаграмма классов

3.3. Пример построения диаграммы деятельности

Данная диаграмма показывает переходы между видами деятельности. Рассмотрим прецедент Заказ стандартного комплекта. Клиент открывает новое окно заказа. Система выводит список устройств, составляющих стандартный комплект и отображает форму покупки. Если пользователь не начинает ввод данных (фамилия, адрес и т.д.) и истекает установленное время ожидания, то выполнение прецедента завершается. Если клиент вводит данные, то система проверяет их. При обнаружении неполных данных происходит возврат в состояние *Отобразить форму покупки*. Если данные введены полностью,

система сохраняет заказ и отправляет данные о нем (номер, дата и т.д.) клиенту по электронной почте.

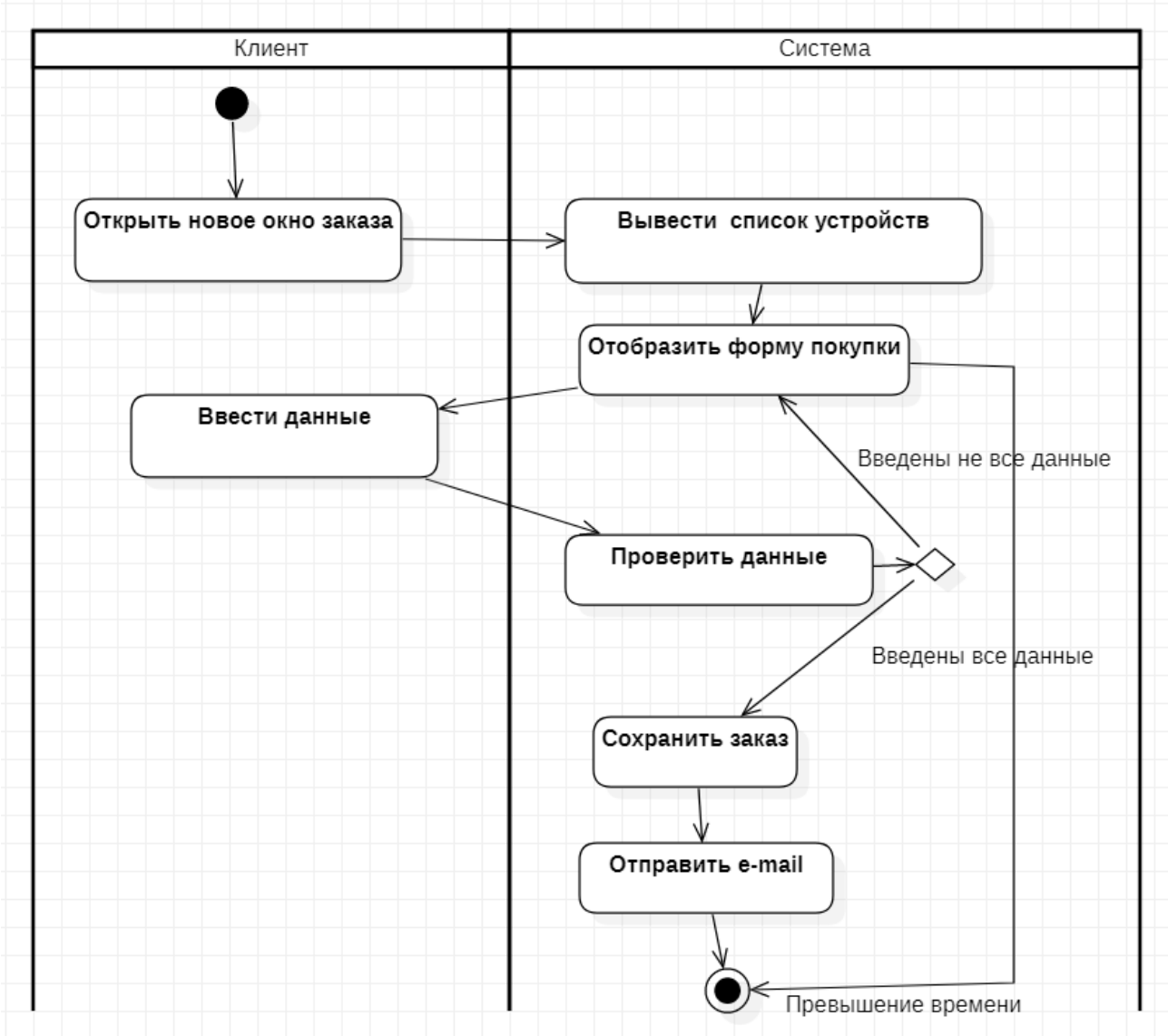

Рис. 2.3. Диаграмма деятельности

3.4. Пример построения диаграммы последовательностей

Построим диаграмму последовательностей для прецедента *Заказ комплекта оборудования* (рис.2.4). Стрелки представляют сообщение, направляемое от вызывающего объекта к операции (методу) вызываемого объекта. Стрелка, указывающая на объект-получатель, предполагает автоматический возврат управления отправителю, поэтому показывать на диаграмме возврат необязательно. Внешний субъект – *Клиент* (на диаграмме последовательностей приведен стандартным графическим обозначением, в виде фигуры «человечка») принимает решение об отображении окна комплектования.

Сообщение *Открыть новое* отправляется объекту класса *Окно комплектования*. В результате создается новое *Окно комплектования*. Объекту *Окно комплектования* необходимо отобразить себя вместе с данными, относящимися к комплекту. С этой целью отправляет сообщение объекту *Комплект*, объект *Комплек*т – объекту *Устройство*. Сообщение *Подтвердить комплект* вызывает отправку сообщения *Подготовить для заказа* объекту *Заказ*. Это приводит к созданию временного объекта *Заказ*, отображаемого в объекте-окне *Окно заказа*. В ответ на принятие клиентом условий заказа (*Подтвердить заказ*) объект *Окно заказа* инициирует создание постоянного заказа (*Сохранить заказ*). После этого объект-заказ *Заказ* устанавливает связь с заказанным комплектом (*Комплект*), а также клиентом (*Клиент – внутренняя сущность*). После того, как эти объекты связываются в базе данных, объект *Заказ* отправляет электронное сообщение внешнему субъекту-клиенту. Клиент является по отношению к системе одновременно и внешней, и внутренней сущностью. Он представляет собой внешнюю сущность, т.к. взаимодействует с системой извне. Однако информация о клиенте должна храниться в системе, чтобы иметь возможность установить идентичность внешнего клиента и внутренней сущности.

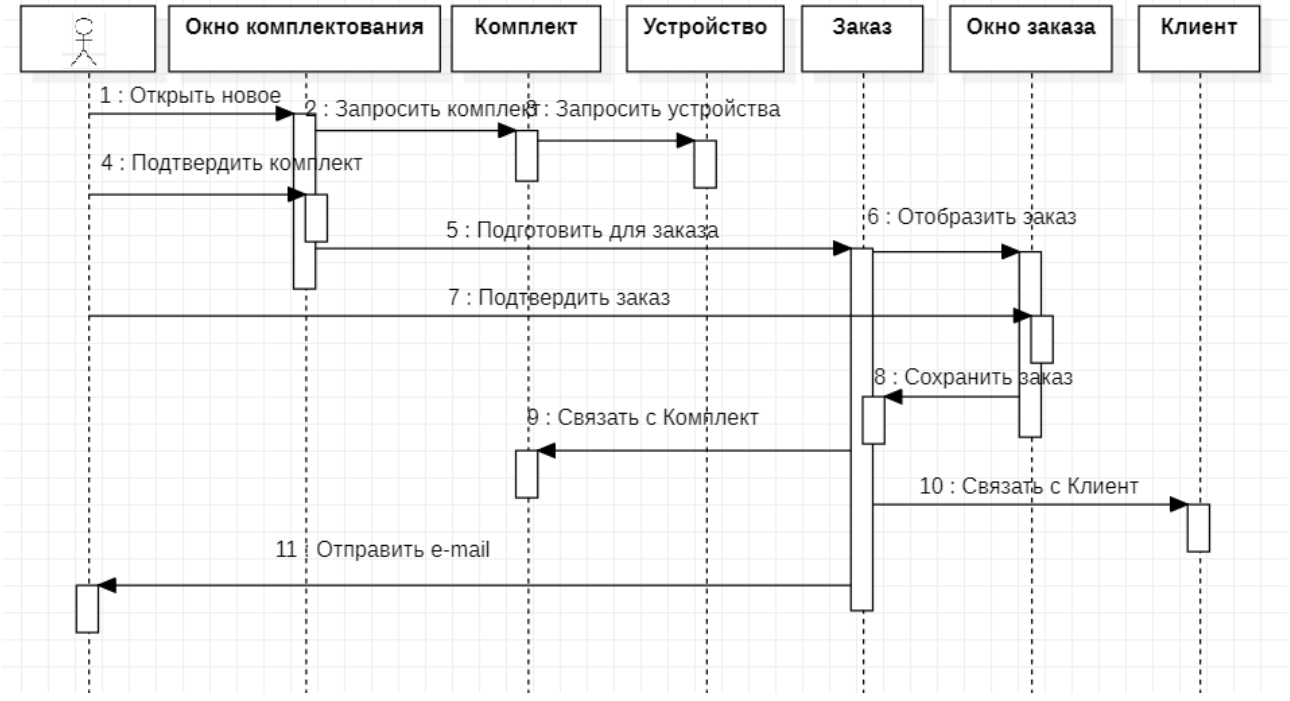

Рис. 2.4. Диаграмма последовательностей

Отметим, что исследование взаимодействий между классами приводит к выявлению операций. После построения диаграмм последовательностей можно вернуться к диаграмме классов и у каждого класса уточнить операции. (Пока на диаграмме классов мы третью строчку для операций оставили пустой).

3.5. Пример построения диаграммы состояний

Диаграмма состояний для каждого состояния объекта определяет действие, выполняемое объектом при получении им сигнала о событии. Начальное состояние объекта класса – *Новый заказ*. Это одно из вложенных состояний составного состояния *Ожидающий заказ*. К другим относятся состояния *Невыполненный заказ* и *Предстоящий заказ*. Из каждого из этих состояний возможны два перехода: в состояние *Отмененный заказ* и *Готов к отгрузке*. Рядом с каждой из стрелок прописаны условия перехода из одного состояния в другое.

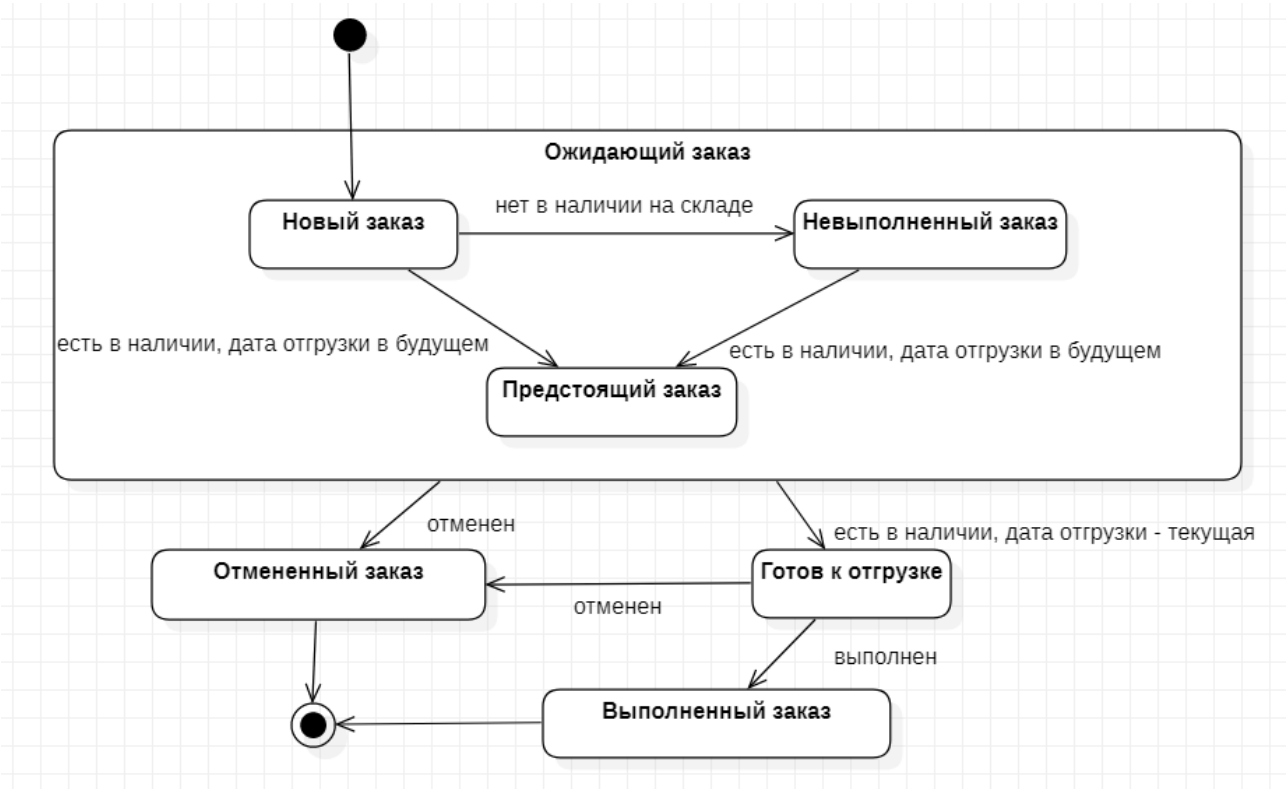

Рис. 2.5. Диаграмма состояний

4. ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

По данным описаниям построить:

а) диаграмму прецедентов;

б) диаграмму классов;

в) диаграмму деятельности (для одного или нескольких прецедентов);

г) диаграмму последовательностей (для одного или нескольких прецедентов);

д) диаграмму состояний (для одного объекта).

Вариант 1. Модуль «Школьная библиотека».

Примерное описание: Читатели (ученики и работники школы) получают книги в библиотеке для временного пользования. В каталоге книги распределены по разделам: учебная – по разным классам, методическая – по разным предметам, художественная литература – по классам и общая. На каждого читателя оформляется электронный формуляр. Необходимо вести учет книг библиотеки, читателей, список выданных книг, предусмотреть возможность напоминания через электронную почту о дате возврата книги, возможность пополнения каталога, оформления отчетов, печати нужных документов.

Вариант 2. Модуль «Прием в вуз».

Примерное описание: При поступлении в вуз абитуриенты подают заявление в электронной форме, где указываются персональные данные (фамилия, имя, отчество, адрес, дата рождения и т.д.), данные об обучении в учебных заведениях, номер аттестата или диплома. Прием в вуз идет по результатам ЕГЭ. На творческие специальности возможен дополнительный экзамен. Необходимо вести учет абитуриентов, предусмотреть возможность отправки запроса в федеральную базу для получения результатов ЕГЭ, составление списков абитуриентов по сумме баллов, возможность отправки электронных писем абитуриентам с напоминаниями о предоставлении оригиналов, печать отчетов и требуемых документов.

Вариант 3. Модуль «Подготовительные курсы в вуз».

Примерное описание: При вузе организованы подготовительные курсы. Абитуриенты выбирают нужные дисциплины из предложенного списка, производят оплату. С абитуриентом составляется договор. При оплате курса, состоящего из нескольких дисциплин, предусмотрены скидки. Дисциплины закрепляются за преподавателями. Необходимо вести учет абитуриентов, составленных договоров, поступивших оплат, списков групп, закрепленных за преподавателем дисциплин, предусмотреть возможность предупреждения студентов по электронной почте при переносе или отмене занятий, возможность оформления и печати отчетов.

Вариант 4. Интернет-магазин по продаже книг.

Примерное описание: Клиент выбирает книгу на Web-странице магазина. Он оформляет заказ. Система должна проверить наличие выбранных книг и оформить счет к оплате. Постоянным клиентам и/или при крупном заказе предусмотрены скидки. После проведения оплаты система должна проверить поступление оплаты и оформить необходимые документы для доставки книги. Необходимо вести учет клиентов, книг, поступивших оплат, оформленных и выполненных заказов.

Вариант 5. Интернет-магазин по продаже обучающих программ.

Примерное описание: Клиент выбирает обучающую программу на Web-странице магазина. Он оформляет заказ. Система должна оформить счет к оплате. Постоянным клиентам и/или при крупном заказе предусмотрены скидки. После проведения оплаты система должна проверить поступление оплаты и предоставить возможность скачивания программы. Необходимо вести учет клиентов, поступивших оплат, оформленных и выполненных заказов.

Вариант 6. Модуль «Внешкольный учебный центр».

Примерное описание: Внешкольный учебный центр предоставляет возможность пройти обучение на платных курсах. Обучающиеся выбирают предметы, вносят оплату. Определенная часть оплаты перечисляется преподавателю. Необходимо вести учет обучающихся, внесенных оплат, закрепления курсов за преподавателями, проведенных занятий, предусмотреть

возможность оформления и печати отчетов, предупреждения обучающихся по электронной почте об отмене/переносе занятий.

Вариант 7. Модуль «Кафедра»

Примерное описание: Преподаватели кафедры могут вести занятия на нескольких факультетах вуза. Преподавание ведется по учебному плану, составленному для каждого направления подготовки. Учебные дисциплины могут входить в обязательную часть или вариативную (профильную), устанавливаемую вузом. Каждая дисциплина закрепляется за преподавателем. Необходимо вести учет нагрузки преподавателей, распределения дисциплин, закрепления студентов за научными руководителями при выполнении курсовых и дипломных работ, своевременного возвращения выданной студентам методической литературы, предусмотреть возможность оформления отчетов.

Вариант 8. Модуль «Деканат»

Примерное описание: После приема в вуз в деканат передаются личные дела первокурсников, содержащих персональные данные (фамилия, имя, отчество, дата рождения, адрес и т.д.). В деканате происходит распределение по группам. Необходимо вести учет успеваемости, посещаемости студентов. Модуль должен хранить результаты экзаменов по курсам и семестрам, пройденные дисциплины. Предусмотреть возможность автоматического формирования и печати ведомостей и справок, учета выданных документов.

Вариант 9. Модуль «Школьный психолог»

Примерное описание: Школьный психолог проводит индивидуальные занятия со школьниками и тестирования. Индивидуальные занятия проводятся для желающих по записи. Тестирование проходит для всего класса по определенному графику. Модуль должен хранить данные об учениках (персональные данные, данные о родителях), характеристики, составленные психологом, тесты, результаты тестирования, отчеты о проведенных индивидуальных занятиях. Предусмотреть возможность оформления и печати отчетов, возможность уведомления классного руководителя и родителей в необходимых случаях, возможность составления или импорта новых тестов.

Вариант 10. Модуль «Бухгалтерия: Зарплата работников школы»

Примерное описание: Каждый из работников школы имеет табельный номер. Работниками являются учителя, администрация, технические работники. Зарплата зависит от разряда или квалификации, нагрузки (для учителей и членов администрации – в часах за неделю), отработанных дней, наличия/отсутствия больничных. Могут быть надбавки к зарплате за оказанные дополнительные платные услуги. Необходимо вести учет работников, расчет зарплаты, налоговых отчислений, формировать для каждого работника листок с данными о зарплате.

Вариант 11. Модуль «Дополнительные платные занятия».

Примерное описание: Вуз предоставляет возможность пройти обучение на дополнительных платных курсах. Обучающиеся выбирают дисциплины, вносят оплату. Определенная часть оплаты перечисляется преподавателю. Необходимо вести учет обучающихся, внесенной оплаты, закрепления курсов за преподавателями, проведенных занятий, предусмотреть возможность оформления и печати отчетов, предупреждения обучающихся по электронной почте об отмене/переносе занятий.

Вариант 12. Модуль «Школьное питание».

Примерное описание: В школьной столовой предусмотрены комплексные обеды. Имеется список продуктов. Из данных продуктов по рецептам приготавливаются блюда. Модуль должен хранить информацию о продуктах (наличие/отсутствие, пищевая ценность – белки, жиры углеводы, калорийность, цена), рецепты, сформированные комплексные обеды, возможность автоматического расчета пищевой ценности и стоимости блюда, комплексного обеда. Предусмотреть возможность ввода новых рецептов, печати меню, нужных документов и отчетов.

Вариант 13. Модуль «Классный руководитель».

Примерное описание: У классного руководителя хранится информация об учениках (фамилия, имя, отчество, дата рождения, адрес, посещаемые кружки и секции), о родителях (фамилия, имя, отчество, адрес, место работы, контактные телефоны и т.д.). Модуль должен хранить информацию об успеваемости

учеников, о посещении занятий, данные об участии в мероприятиях, информацию о проведенных мероприятиях в классе, характеристики учеников. Предусмотреть возможность оформления и печати нужных документов, возможность информирования о предстоящих/проведенных мероприятиях родителей по электронной почте.

Вариант 14. Модуль «Заместитель директора школы»

Примерное описание: Заместитель директора школы отвечает за учебный процесс, успеваемость учеников и посещение ими занятий. Распределяет нагрузку учителей (закрепление предметов за учителями). Необходимо вести учет нагрузки учителей, учет проведенных уроков, учет оценок, пропущенных уроков по классам, по параллелям, по школе. Предусмотреть возможность автоматического формирования и печати отчетов, содержащих данные о средних оценках по предмету, о средней оценке у каждого учителя, по классам.

Вариант 15. Модуль «Административно-хозяйственная часть дошкольного учреждения»

Примерное описание: Заведующий административно-хозяйственной частью (АХЧ) производит закупку материальных средств. Возможна закупка мебели, оргтехники, инструментов, книг, игрушек, моющих средств и др. Каждый закупленный товар закрепляется за работником учреждения. Возможно, что оргтехника передается другому работнику или проводится ремонт. Необходимо вести учет закупленных средств, их стоимости, проведенного ремонта, материально-ответственных лиц. При окончании срока службы производится списание средства.

Вариант 16. Модуль «Административно-хозяйственная часть школы»

Примерное описание: Административно-хозяйственная часть школы производит закупку материальных средств. Возможна закупка школьной мебели, оргтехники, компьютеров, книг, моющих средств. Каждый закупленный товар закрепляется за работником учреждения. Мебель, компьютеры, оргтехника дополнительно закрепляются за определенным кабинетом. Возможна их перестановка, которая должна учитываться у заведующего

административно-хозяйственной частью. Необходимо вести учет закупленных средств, их стоимости, материально-ответственных лиц, состояния, проведенного ремонта. По окончанию срока службы производится списание средства.

Вариант 17. Модуль «Школьный медпункт»

Примерное описание: Школьный врач/медсестра для каждого школьника ведет карточку, куда вводятся данные о состоянии здоровья, проведенных медосмотрах, антропометрические данные (рост, вес и пр.), выполненные прививки, перенесенные заболевания, наличие/отсутствие аллергии. Необходимо вести учет всех перечисленных данных, предусмотреть информирование родителей и классного руководителя, выдачу направления к врачу-специалисту в поликлинику, оформление справки о состоянии здоровья, сертификата о прививках, учет выданных справок.

Вариант 18. Модуль «Спортивные секции»

Примерное описание: В спортивном комплексе ведется учет студентов, посещающих секции. Имеются платные и бесплатные занятия. За бесплатные занятия деньги перечисляет вуз, в котором учится студент. Для этого студент должен предъявить студенческий билет. Определенный процент поступившей оплаты перечисляется в виде надбавки преподавателям-тренерам. Для постоянных клиентов предусмотрены скидки. Необходимо вести учет студентов, занятость спортивных залов, проведенных занятий, руководителей секций, внесенной оплаты.

Вариант 19. «Музыкальная школа»

Примерное описание: В музыкальной школе несколько отделов: фортепиано, струнных инструментов, духовых инструментов. Ученики по специальности (инструмент) прикрепляются к отдельному учителю, занятия проходят индивидуально. По таким дисциплинам, как хор и сольфеджио, занимаются в группе. Необходимо вести учет успеваемости и посещения учеников, результатов сдачи экзаменов, учет предметов, которые ведутся

конкретными учителями. Предусмотреть возможность информирования родителей через электронную почту о предстоящих собраниях и концертах.

Вариант 20. Модуль «Курсы по подготовке к школе».

Примерное описание: При школе организованы курсы для будущих первоклассников. Родители вносят оплату, с ними составляется договор. Предметы закрепляются за учителями. Необходимо вести учет обучающихся, посещения занятий, результатов выполненных контрольных работ и тестов, договоров, поступивших оплат, списков групп, закрепленных за учителем предметов, предусмотреть возможность отправки сообщений по электронной почте при переносе или отмене занятий, возможность оформления и печати отчетов.

Вариант 21. Модуль «Прокат спортивного инвентаря».

Примерное описание: В спорткомплексе вуза осуществляется прокат спортивного инвентаря. Студенты данного вуза могут пользоваться услугами проката бесплатно. Посторонние клиенты могут зарегистрироваться через сайт или подойти лично. Для постоянных клиентов предусмотрены скидки. Необходимо вести учет клиентов, их персональных данных, имеющегося в наличии и выданного инвентаря, поступивших оплат, пожеланий клиентов. Предусмотреть возможность отправки сообщения по электронной почте о сроках возврата, о новых услугах, возможность формирования и печати отчетов.

Вариант 22. Модуль «Документация школы».

Примерное описание: Администрация школы хранит персональные данные учеников и их родителей, учебные планы по классам и календарные планы по предметам. Ввод информации осуществляет делопроизводитель. Данные об учениках может просматривать классный руководитель. Учебные планы для просмотра доступны всем. Календарные планы по предметам доступны для просмотра учителям-предметникам. Необходимо предусмотреть хранение всех введенных данных, возможность формирования и печати списков, отчетов, возможность отправки сообщения учителям при обнаружении неточностей в документах.

Вариант 23. Интернет-тестирование школьников.

Примерное описание: По графику управления образования проходит тестирование школьников по основным предметам. Для тестирования выделяется определенное время, всем участникам отправляются логины и пароли. Необходимо вести учет вопросов тестов, самих тестов, данных об участниках, результатов тестирования, определения средних баллов школьников по школам, классам, предметам. Предусмотреть возможность формирования и печати отчетов, отправки сообщения школьникам и школам результатов тестирования по электронной почте.

Вариант 24. Модуль «Студенческое общежитие».

Примерное описание: По направлению деканата в общежитие приходят студенты. Комендант расселяет их по свободным комнатам, по возможности учитывая, на каком факультете и курсе обучается студент. Проживающие могут подать заявку об устранении неисправностей оборудования. Необходимо вести учет комнат (этаж, площадь и т.д.), учет проживающих, их персональных данных, поступивших оплат за проживание, выявленных нарушений, заявок на ремонт оборудования или мебели. Предусмотреть возможность формирования и печати отчетов, отправки данных о свободных местах в деканаты.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Проектирование информационных систем Автор: В.И. Грекул. Национальный Открытый Университет «ИНТУИТ». https://intuit.ru/studies/courses/1178/330/info

2. Программная инженерия 2021. Автор: В.Тарасенко https://stepik.org/course/80872/promo

3. Мацяшек, Лешек А. Анализ требований и проектирование систем: Разработка информационных систем с использованием UML / Лешек А. Мацяшек; Маккуар. ун-т, Сидней, Австралия. – М. : Вильямс, 2002. – 428 с.

4. Визуальное моделирование: теория и практика Автор: Д.В. Кознов. Национальный Открытый Университет «ИНТУИТ» https://intuit.ru/studies/courses/1041/218/info

5. Введение в UML Автор: А.В. Бабич. Национальный Открытый Университет «ИНТУИТ» https://intuit.ru/studies/courses/1007/229/info

6. Леоненков, А. В. Самоучитель UML: Самоучитель / Леоненков А.В., – 2-е изд., перераб. и доп. – СПб:БХВ-Петербург, 2015. – 418 с. ISBN 978-5-9775-1216-9. – Текст : электронный. – URL: https://znanium.com/catalog/product/939591

7. Мацяшек, Л. А. Практическая программная инженерия на основе учебного примера : монография / Л. А. Мацяшек, Б. Л. Лионг. – 4-е изд. – Москва : Лаборатория знаний, 2020. – 959 с.

8. Проектирование архитектур информационных систем : методические указания к лабораторным работам/ сост. К. С. Беляев. – Ульяновск : УлГТУ, 2010. – 48 с.

9. Анализ требований к автоматизированным информационным системам. Автор: Ю.А. Маглинец. Национальный Откр. Ун-т «ИНТУИТ» https://intuit.ru/studies/courses/2188/174/info

10. Гагарина, Л. Г. Разработка и эксплуатация автоматизированных информационных систем: Учебное пособие / Гагарина Л.Г. - Москва :ИД ФОРУМ, НИЦ ИНФРА-М, 2017. - 384 с. (Профессиональное образование) ISBN

978-5-8199-0316-2. – Текст : электронный. – URL: https://znanium.com/catalog/product/612577

11. Заботина, Н. Н. Проектирование информационных систем: Учебное пособие / Н.Н. Заботина. - Москва : НИЦ ИНФРА-М, 2014. – 331 с. + (Доп. мат. znanium.com). – (Высшее образование: Бакалавриат). ISBN 978-5-16-004509-2. – Текст : электронный. - URL: https://znanium.com/catalog/product/454282

12. Разработка информационных систем в образовании с использованием UML: учеб.-метод. пособие / авт.-сост. Ч.Б.Миннегаливеа. – Казань: Казан.ун-т, 2012. – 48 с.

13. Современные технологии программирования. А.П. Пашкевич, О.А. Чумаков. Белорус.гос.ун-т информатики и радиоэлектроники. Минск, 2007

14. Анализ требований к автоматизированным информационным системам. Автор: Ю.А. Маглинец. Национальный Откр. Ун-т «ИНТУИТ» https://intuit.ru/studies/courses/2188/174/info

15. Ларман, Крэг. Применение UML и шаблонов проектирования. Введение в объектно- ориентированный анализ и проектирование / Крэг Ларман; Пер. с англ. – М.: Издат. дом «Вильямс», 2013. – 736 с.

16. Мартин Фаулер. UML. Основы: краткое руководство по стандартному языку объектного моделирования : Символ-Плюс, 2002, 185 с.