Объектно-ориентированный анализ и проектирование информационной системы

Методические указания к лабораторной работе.

1. Введение в объектно-ориентированный анализ

Объектно-ориентированный подход завоевал широкое распространение в области проектирования информационных систем как мощный и эффективный инструмент архитекторов и разработчиков программного обеспечения. Этому способствовал тот факт, что выделение в окружающем мире обособленных объектов и их взаимодействий является наиболее естественным для человека способом познания мира. В настоящее время можно говорить о целой объектно-ориентированной методологии создания программных систем, включающей языки моделирования (такие как UML – Unified Modelling Language), поддерживающие их средства автоматизированного проектирования (CASE-средства), объектно-ориентированные языки программирования, а также методики применения указанных средств для достижения искомого результата.

Обычно при создании программной системы с помощью объектно-ориентированной методологии выделяют три основных этапа:

- этап объектно-ориентированного анализа (OOA – Object-Oriented Analysis)
- этап объектно-ориентированного проектирования (OOD – Object-Oriented Design)
- этап объектно-ориентированного программирования (OOP – Object-Oriented Programming)

На этапе OOA происходит осмысление предметной области в тесном взаимодействии разработчиков и заказчиков системы. Результатами данного этапа являются модели предметной области и вариантов использования. На этапе OOD полученные знания о предметной области проецируют на будущую программную реализацию. Результатами этого этапа являются, в частности, модели классов, компонентов и взаимодействия. Наконец, на этапе OOP проект превращается в конкретную программную реализацию. В некоторых случаях на этом этапе возможны определенные изменения и в исходном дизайне.

2. UML как важнейший инструмент объектно-ориентированного анализа и проектирования

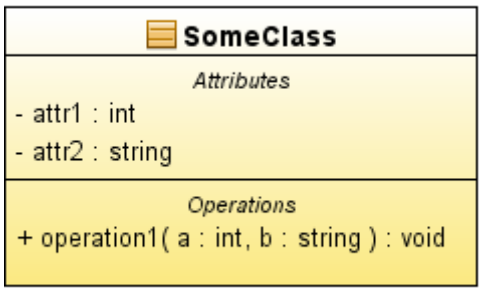
Язык UML был создан с целью унификации множества графических нотаций, созданных для визуализации артефактов объектно-ориентированного анализа и проектирования. В итоге был получен язык, понятный всем грамотным разработчикам и архитекторам программного обеспечения. Успех UML позволил в настоящее время расширить области его применения и за пределы области разработки программных систем.

Язык UML включает графические модели, или диаграммы, различных типов. Некоторые из них служат для моделирования статической структуры программной системы, например, диаграмма классов, диаграмма компонентов, диаграмма развертывания. Другие модели используются для моделирования динамических аспектов, или поведения, программных систем: диаграмма последовательностей, диаграмма состояний и т.д. В данном

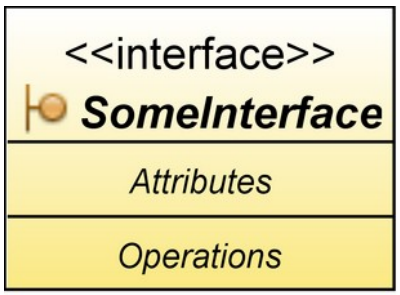
пособии рассмотрим два наиболее базовых типа диаграмм: диаграмма классов и диаграмма последовательностей.

2.1. Диаграмма классов UML

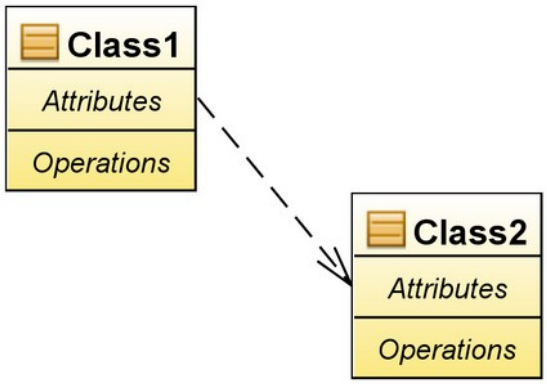
Основным компонентом диаграммы является **класс**. Класс представляется прямоугольником, разделенным на три зоны. Верхняя зона содержит название класса. Затем идут зона атрибутов класса и зона операций класса.



Классы могут иметь стереотипы. Например, класс со стереотипом <<interface>> соответствует понятию интерфейса в языке программирования Java.

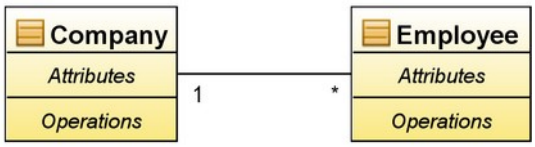


Классы могут состоять в отношениях. Простейшим отношением является зависимость (dependency). Это отношение означает, что одному классу необходим другой класс для выполнения своих функций. В UML зависимость отражается пунктирной стрелкой.

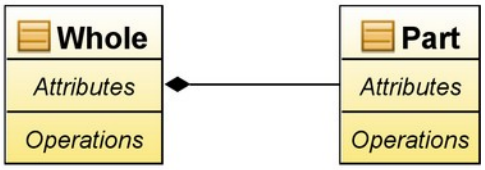


Более общим отношением является ассоциация (association). Это отношение указывает логические взаимосвязи сущностей. Например, между классами Company и Employee может быть установлена ассоциация «один ко многим», означающая, что в компании может работать много сотрудников, но каждый сотрудник работает только в одной компании. В UML ассоциация отображается линией без стрелок. На концах ассоциации

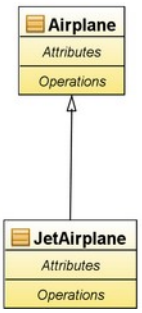
может указываться множественность отношения (0, 1, n, * и т. д.)



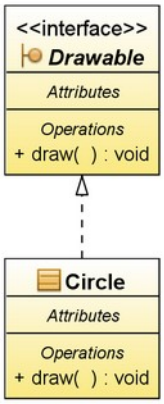
Существуют различные типы ассоциаций. Важным частным случаем является композиция (Composition). Это частный случай отношения типа «целое — составная часть», или агрегирования. Например, автомобиль состоит из деталей, группа состоит из студентов и т.д. Одним из признаков композиции в программной системе является ситуация, когда объект одного класса создает экземпляры объектов другого класса, хранит их и удаляет при своем удалении. В UML композиция обозначается линией с закрашенным ромбом, расположенным около класса, выступающего в роли целого.



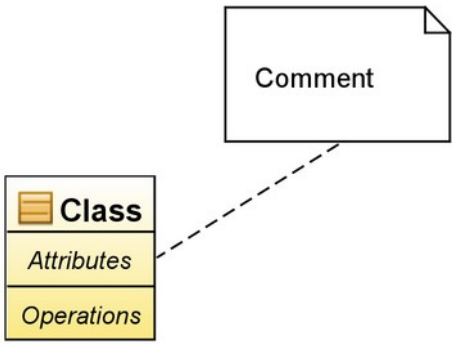
Еще одним важнейшим отношением между классами является обобщение (generalization) или, иначе говоря, наследование. В UML обобщение обозначается линией с незакрашенной стрелкой, идущей от частного класса к общему (или от потомка к родителю).



Отношение обобщения может существовать и между интерфейсами. Однако, между классом и интерфейсом обобщения быть не может. Вместо этого используется родственное отношение реализации (implementation), обозначаемое в UML пунктирной линией с незакрашенной стрелкой, идущей от класса к интерфейсу.

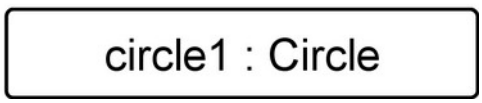


Некоторые неформальные аспекты диаграммы классов могут быть обозначены комментариями.

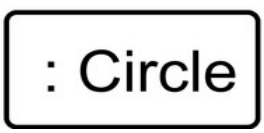


2.2. Диаграмма последовательностей UML

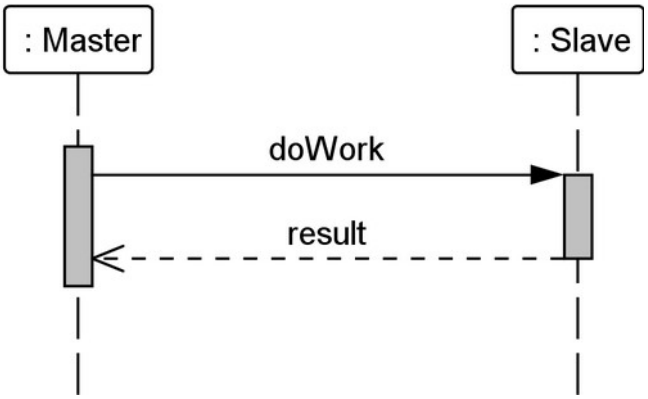
В отличие от диаграммы классов диаграмма последовательностей отражает не структуру, а поведение программной системы. Поэтому вместо классов на этой диаграмме отображаются объекты — конкретные экземпляры классов. Объект обозначается прямоугольником, в котором указывается имя объекта и имя класса через двоеточие.



Имя объекта можно опускать. Такой безымянный объект следует понимать, как «какой-либо объект указанного класса».

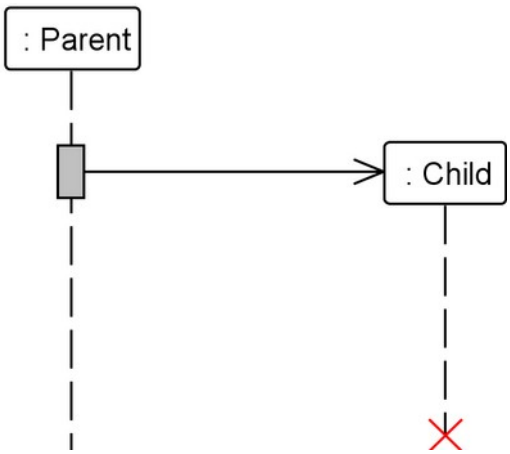


С каждым объектом ассоциируется линия жизни (lifeline). Линия жизни идет сверху вниз, отражая последовательность событий в ходе выполнения программной системы. Основной элемент диаграммы последовательностей — это сообщения, которыми обмениваются объекты. Сообщение обозначается стрелкой от линии жизни источника сообщения к линии жизни получателя. На практике сообщением может быть, например, вызов одним объектом метода другого объекта. Тот факт, что отправка сообщения является результатом деятельности объекта, а прием сообщения связан с его обработкой показывается утолщением линии жизни в виде серого прямоугольника.



Около линии сообщения, как правило, подписывается название сообщения, которое совпадает с названием операции объекта-получателя, вызываемой сообщением. Синхронные сообщения подразумевают как запрос, так и ответ. Ответное сообщение отображается пунктирной стрелкой, над которой может быть подписан результат операции.

Не все объекты существуют в течение одинакового времени. Тот факт, что объект может создать другой объект отражается специальным сообщением создания (Create message). Удаление объекта отображается крестом в конце его линии жизни.



Диаграммы последовательностей могут использоваться для визуализации достаточно сложных взаимодействий объектов, включающих условия и циклы.

3. Выполнение объектно-ориентированного анализа

Задачей объектно-ориентированного анализа является переход от неформального описания задания к формализованному. В рамках данных методических рекомендаций будем использовать две формы формализованного описания: модель предметной области и

варианты использования.

3.1. Модель предметной области

Модель предметной области строится в форме диаграммы классов UML, однако, важно отличать классы предметной области от классов, которые используются в программной реализации системы. Фактически, классы предметной области — это просто отражение понятий реального мира и их взаимосвязей. Они не имеют прямого отношения к программным классам.

Первая задача, которую необходимо решить при формировании модели предметной области, - это определение состава понятий. Список понятий предметной области может быть достаточно широким. Не стоит бояться включать в модель предметной области новые сущности, если это помогает прояснить задачу.

Для получения начального списка понятий, от которого можно отталкиваться в дальнейшем анализе, рекомендуем использовать простой способ на основе выделения существительных. Допустим, словесная формулировка задания звучит следующим способом:

«Разработать сервис записной книжки, которая хранит имена людей и их адреса e-mail. Сервис должен позволять добавлять новые контакты в книжку и находить контакты по имени».

Найдем и подчеркнем в этом описании все существительные (возможно, вместе с уточняющими их определениями):

*«Разработать **сервис** **записной книжки**, которая хранит **имена** **людей** и их **адреса e-mail**. **Сервис** должен позволять добавлять новые **контакты** в **книжку** и находить **контакты** по **имени**».*

Итак, получаем первоначальный список понятий:

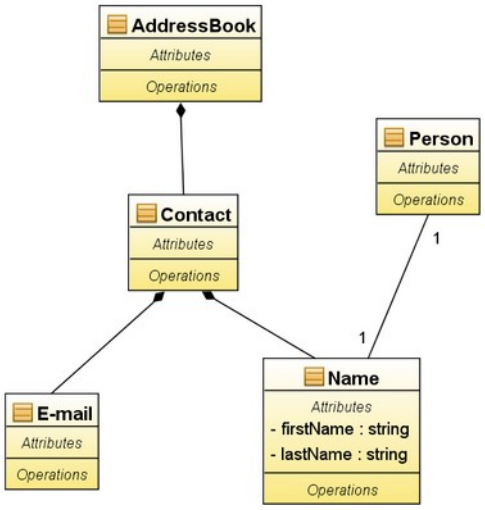
- сервис
- записная книжка
- имя
- человек
- адрес e-mail
- контакт

На этом этапе стоит выбросить из списка понятия типа «сервис» или «система», т.к. они характеризуют скорее не саму предметную область, а то, что нам следует создать, и чего пока еще не существует.

Размещаем понятия на диаграмме классов. Думаем, не надо ли включить еще какие-то понятия? В данном случае, в силу простоты задания, полученного списка, по всей видимости, достаточно.

Отражаем в модели отношения между понятиями. В модели предметной области, в первую очередь, выделяют ассоциации. Видно, что записная книжка состоит из контактов — это композиция. Контакт содержит имя и адрес e-mail — это также композиция. Имя ассоциируется с человеком, причем можно наложить ограничение, что эта ассоциация имеет тип «один к одному» (хоть это и некоторое упрощение реальной ситуации). Имя является сложным понятием, поэтому соответствующий класс имеет атрибуты: firstName и lastName.

Окончательная модель предметной области выглядит следующим образом.



3.2. Варианты использования

Варианты использования описывают возможные действия пользователя и реакцию системы на эти действия. В UML имеется отдельный тип диаграммы вариантов использования (Use Case Diagram). Однако, мы рекомендуем использовать не графическое, а текстовое описание вариантов использования.

Каждый вариант использования включает название и последовательность действий пользователя и реакций системы, возможно, с альтернативами. Варианты использования должны охватывать основные различимые пользователем сценарии взаимодействия с системой. Сразу поясним соотношение понятий «вариант использования» и «сценарий»: это отношение является композицией. Вариант использования состоит из множества сценариев (в частном случае — из одного сценария) его реализации.

Исходя из текстового описания задания можно выделить два основных варианта использования: Добавление контакта и Поиск контакта. Опишем эти варианты использования.

Добавление контакта

1. Пользователь входит в Систему и вводит имя человека и его адрес e-mail.
2. Система проверяет наличие контакта с данным именем
3. Если контакт не найден, система создает новый контакт и добавляет его в записную книжку.

Альтернатива

3а. Если контакт найден, система выдает сообщение об ошибке и не изменяет записную книжку.

Примечание. Вариант использования — это не догма, а результат согласования видения системы заказчиками и разработчиками. Чем точнее описаны варианты использования, тем меньше будет недоразумений при сдаче системы. В частности, здесь договорились, что

система не добавляет записи с существующим именем. Можно было бы предусмотреть возможность в этом случае заменить старую запись на новую и т.д.

Поиск контакта

1. Пользователь входит в Систему и вводит имя человека
2. Система проверяет наличие контакта с данным именем
3. Если контакт найден, система выдает адрес e-mail из этого контакта.

Альтернатива

3а. Если контакт не найден, система выдает пользователю соответствующее сообщение.

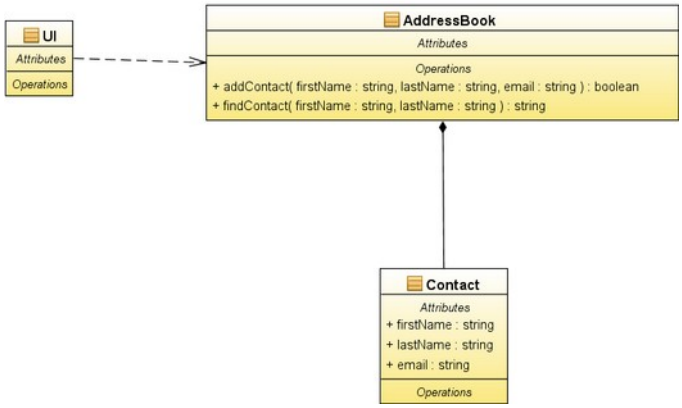
Итак, имеем два варианта использования, каждый из которых имеет два вложенных сценария (определяемых альтернативной последовательностью).

4. Выполнение объектно-ориентированного проектирования

Объектно-ориентированное проектирования выполняется с целью перехода от модели предметной области и вариантов использования к формализованной модели будущей программной системы. На данном этапе ограничимся двумя формами такой модели: диаграмма классов проектирования и диаграмма последовательностей.

4.1. Диаграмма классов проектирования

Эта диаграмма классов внешне похожа на модель предметной области, но существенно отличается от нее по сути. Элементами данной модели являются уже не абстрактные понятия, а реальные классы и интерфейсы будущей программной системы. При этом модель предметной области является отправной точкой (для этого она и создавалась) создания диаграммы классов проектирования. Некоторые классы предметной области могут напрямую перейти в классы проектирования. Часть классов предметной области могут либо не иметь программной реализации, либо могут быть реализованы по-другому (например, как атрибуты классов проектирования). Наконец, на этапе проектирования могут появляться новые классы, которые отсутствовали в предметной области, но необходимы для программной реализации.



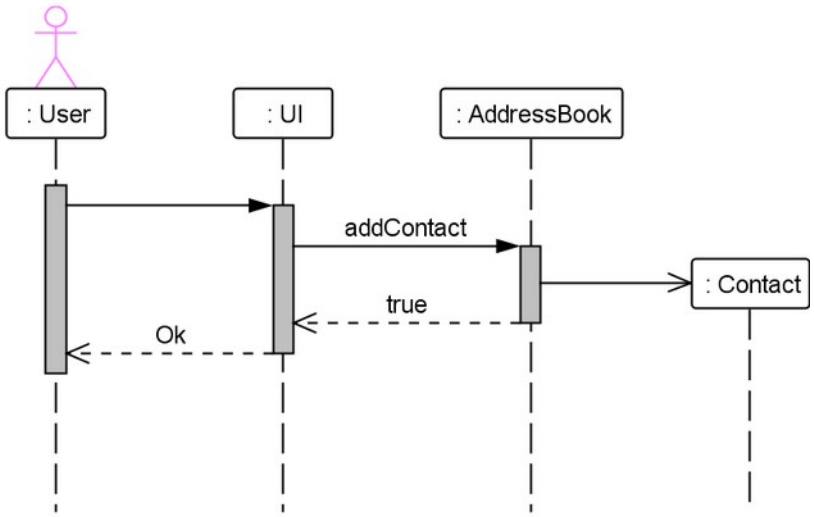
Возвращаясь к нашему примеру, исключаем из состава классов проектирования понятие Человек (Person). С целью получения простой и понятной программной реализации отдельные классы Name и E-mail заменяем соответствующими атрибутами класса Contact. Класс AddressBook переносится в модель проектирования. Возложим на него ответственность за реализацию основных операций (в соответствии с вариантами использования) — добавление addContact и поиск findContact.

Для получения законченной системы необходимо предусмотреть также реализацию пользовательского интерфейса. Возложим эту ответственность на отдельный класс, который назовем UI. В реальности этот класс может олицетворять, например, веб-фронтэнд приложения. Разумеется, полученная диаграмма является минимальной возможной реализацией поставленных требований. В рамках дальнейших лабораторных работ реализация будет дополнена.

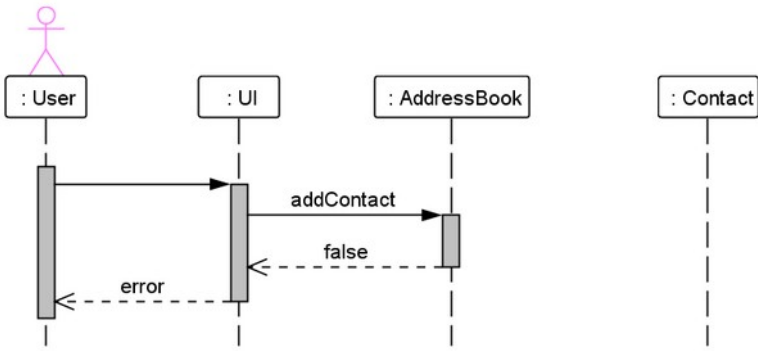
4.2. Диаграмма последовательностей

Несмотря на то, что описание данного вида диаграмм UML идет в тексте после диаграммы классов проектирования, рекомендуется создавать эти модели не по очереди, а параллельно. Отражение взаимодействия объектов на диаграмме последовательностей позволяет конкретизировать, какие именно действия и в какой последовательности выполняют объекты. Именно такой подход позволит понять, какие классы необходимы, а какие — лишние, уже на этапе проектирования.

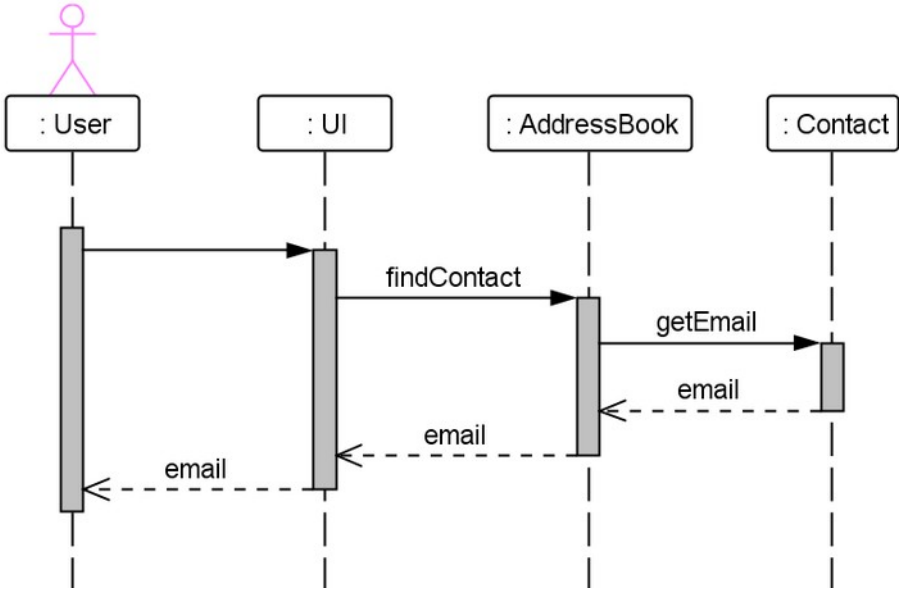
Для каждого сценария, как правило, строится отдельная диаграмма последовательностей. Диаграмма последовательностей для успешного сценария варианта использования Добавление контакта будет выглядеть следующим образом.



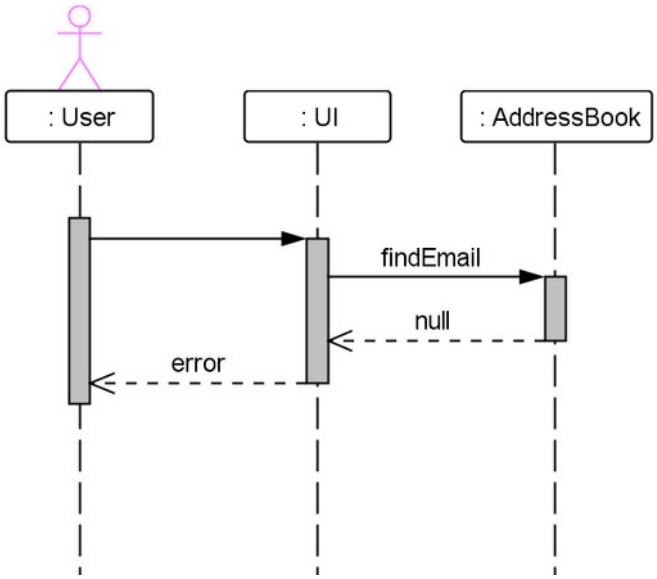
Тот же вариант использования, но альтернативное развитие событий.



Вариант использования Поиск контакта, основной сценарий.



Альтернативный сценарий.



5. Рекомендации по созданию UML диаграмм

UML диаграмма — не самоцель, а способ наглядного представления результатов объектно-ориентированного анализа. В зависимости от ситуации могут использоваться разные способы их создания. На начальном этапе крайне полезно рисовать диаграммы от руки. При этом допускается указывать только действительно важные для понимания аспекты, опуская подробности. Мы рекомендуем, приступая к работе над заданием, начать именно с этого способа.

В дальнейшем, при формировании отчета или документации по программной системе, имеет смысл использовать автоматизированные средства создания UML диаграмм. Существуют как свободно распространяемые средства (например, Dia), так и коммерческие (например, Microsoft Visio). Для выполнения лабораторных работ можно также воспользоваться UML-плагином к среде разработки.

6. Литература

1. Буч Г., Рамбо Д., Якобсон А. Язык UML. Руководство пользователя. ДМК, 2006.
2. Фаулер М. UML. Основы, 3-е издание. Символ-Плюс, 2006.
3. Крэг Ларман. Применение UML 2.0 и шаблонов проектирования, 3-е издание. Вильямс, 2007.