**Лабораторная работа №4 (ЛР4)   
«Объектно-ориентированное моделирование средствами языка UML»**

[Л4.1 Цель работы 1](#_Toc457416583)

[Л4.2 Общие положения по объектному моделированию и языку UML 1](#_Toc457416584)

[Л4.3 Термины и понятия UML 3](#_Toc457416585)

[Л4.3.1 Сущности 3](#_Toc457416586)

[Л4.3.2 Отношения 7](#_Toc457416587)

[Л4.3.3 Диаграммы 9](#_Toc457416588)

[Л4.3.4 Правила и общие механизмы языка UML 11](#_Toc457416589)

[Л4.4 Общее описание характеристик и возможностей CASE-средства проектирования Sparx Enterprise Architect 14](#_Toc457416590)

[Л4.5 Описание интерфейса и основ работы с системой Sparx Enterprise Architect 15](#_Toc457416591)

[*Л4.5.1* Рабочее пространство SEA 15](#_Toc457416592)

[Л4.5.2 Стартовая страница SEA 15](#_Toc457416593)

[Л4.5.3 Главное меню SEA 16](#_Toc457416594)

[Л4.5.4 Создание нового проекта 17](#_Toc457416595)

[Л4.5.5 Шаблоны моделей в SEA 18](#_Toc457416596)

[Л4.6 Описание примеров построения диаграмм 19](#_Toc457416597)

[Л4.6.1 Диаграмма прецедентов 19](#_Toc457416598)

[Л4.6.2 Диаграммы взаимодействия 24](#_Toc457416599)

[Л4.6.3 Диаграмма состояний. 27](#_Toc457416600)

[Л4.6.4 Диаграмма деятельности 29](#_Toc457416601)

[Л4.6.5 Диаграмма классов 31](#_Toc457416602)

[Л4.7 Варианты заданий по ЛР4 32](#_Toc457416603)

[Л4.8 Порядок выполнения ЛР4 33](#_Toc457416604)

[Л4.9 Правила оформления отчета по ЛР4 33](#_Toc457416605)

[Л4.10 Контрольные вопросы по ЛР4 33](#_Toc457416606)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 34](#_Toc457416607)

[ПРИЛОЖЕНИЕ А 35](#_Toc457416608)

[ПРИЛОЖЕНИЕ Б 42](#_Toc457416609)

[ПРИЛОЖЕНИЕ В 46](#_Toc457416610)

## Л4.1 Цель работы

1. Закрепить основные представления об унифицированном языке моделирования UML как средстве проектирования автоматизированных систем.
2. Получить практические навыки построения диаграмм UML в среде CASE средства Sparx Enterprise Architect.

## Л4.2 Общие положения по объектному моделированию и языку UML

При разработке автоматизированных систем и их программного обеспечения наиболее современным и перспективным является *объектно-ориентированный подход.* [1] Согласно ему в качестве основного строительного блока проектируемой системы выступает объект или класс. В самом общем смысле объект - это сущность, обычно извлекаемая из словаря предметной области или решения, а класс является описанием множества однотипных объектов. Каждый объект обладает идентичностью (его можно поименовать или как-то по-другому отличить от прочих объектов), состоянием (обычно с объектом бывают связаны некоторые данные) и поведением (с ним можно что-то делать или он сам может что-то делать с другими объектами).

В качестве примера рассмотрим простую трехуровневую архитектуру биллинговой системы, состоящую из интерфейса пользователя, программно­го обеспечения промежуточного слоя и базы данных. Интерфейс содержит конкретные объекты - кнопки, меню и диалоговые окна. База данных также состоит из конкретных объектов, а именно таблиц, представляющих сущно­сти предметной области: клиентов, продукты и заказы. Программы промежу­точного слоя включают такие объекты, как транзакции и бизнес-правила, а также более абстрактные представления сущностей предметной области (клиентов, продуктов и заказов).

Объектно-ориентированный подход к разработке программного обес­печения является сейчас преобладающим просто потому, что он продемонст­рировал свою полезность при построении систем в самых разных областях любого размера и сложности. Кроме того, большинство современных языков программирования, инструментальных средств и операционных систем яв­ляются в той или иной мере объектно-ориентированными. Объектно-ориентированные методы разработки легли в основу идеологии сборки систем из отдельных компонентов; в качестве примера можно назвать такие технологии, как JavaBeans и СОМ+.

Визуализация, специфицирование, конструирование и документирова­ние объектно-ориентированных систем - это и есть назначение языка UML.

Унифицированный язык моделирования (Unified Modeling Language, UML) является графическим языком для визуализации, специфицирования, конструирования и документирования систем, в которых большая роль при­надлежит программному обеспечению. С помощью UML можно разработать детальный план создаваемой системы, отображающий не только ее концеп­туальные элементы, такие как системные функции и бизнес-процессы, но и конкретные особенности реализации, в том числе классы, написанные на специальных языках программирования, схемы баз данных и программные компоненты многократного использования.

Классы - это самые важные строительные блоки любой объектно-ориентированной системы. Они представляют собой описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой. Класс реализует один или несколько интерфейсов.

В языке UML все сущности подобного рода моделируются как классы. Класс - это абстракция сущностей, являющихся элементами словаря предметной области. Класс представляет не индивидуальный объект, а целую их совокупность. Можно считать, что "стена" - это класс объектов с некоторыми общими свойствами, такими как высота, длина, толщина, несущая это стена или нет, и т.д. При этом конкретные стены будут рассматриваться как отдельные экзем­пляры класса "стена", одним из которых является, например, "стена в юго-западной части моего кабинета".

Графическое изображение класса в UML показано на рис. Л4.1. Такое обо­значение позволяет визуализировать абстракцию независимо от конкретного языка программирования и подчеркнуть ее наиболее важные характеристики: имя, атрибуты и операции. Атрибуты и операции задают сигнатуру класса.

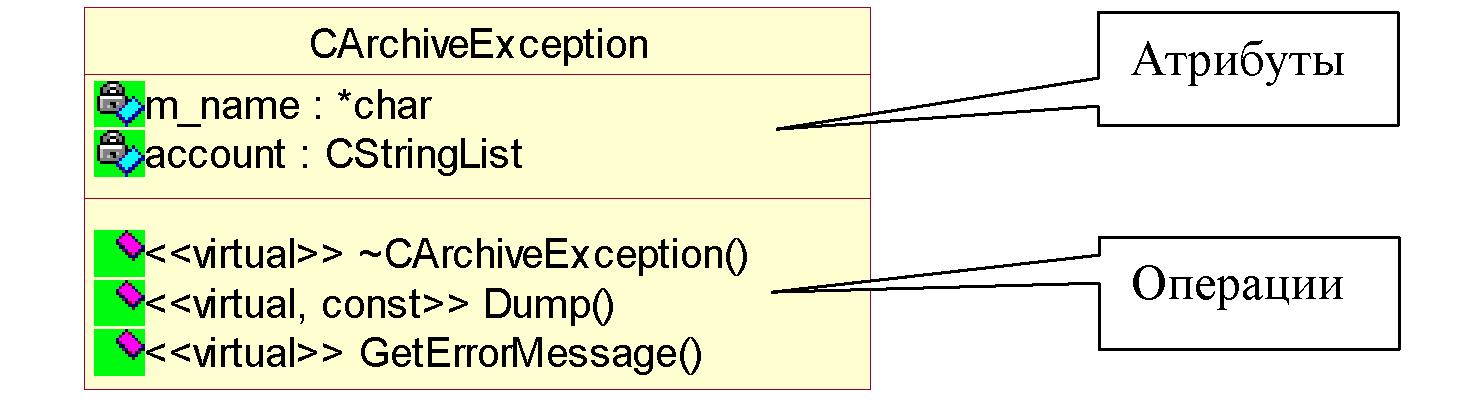


Рис. Л4.1 – Классы

## Л4.3 Термины и понятия UML

Словарь языка UML включает три вида блоков: ***сущности, отношения* и *диаграммы****.*

Л4.3.1 Сущности

*Сущности -* это абстракции, являющиеся основными элементами мо­дели. Отношения связывают различные сущности; диаграммы группируют представляющие интерес совокупности сущностей.

В UML имеется четыре типа сущностей: *структурные, поведенческие, группирующие* и *аннотационные.*

***Структурные сущности*** *-* это имена существительные в моделях на языке UML. Как правило, они представляют собой статические части модели, соответствующие концептуальным или физическим элементам системы. Су­ществует несколько разновидностей структурных сущностей: *Классы, Атрибуты, Операции, Интерфейс, Кооперация, Прецедент, Компонент.*

*Классом* (Class) называется описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой. Графически класс изо­бражается в виде прямоугольника. У каждого класса должно быть имя, отли-чающее его от других классов. Имя класса - это текстовая строка. Взятое са­мо по себе, оно называется простым именем; к составному имени спереди добавлено имя пакета, куда входит класс. Имя класса в объемлющем пакете должно быть уникальным. Имя класса может состоять из любого числа букв, цифр и ряда знаков препинания (за исключением таких, например, как двое­точие, которое применяется для отделения имени класса от имени объемлю­щего пакета). Имя может занимать несколько строк. На практике для имено­вания класса используют одно или несколько коротких существительных, взятых из словаря моделируемой системы. Обычно каждое слово в имени класса пишется с заглавной буквы, например: Customer (Клиент), Wall (Сте­на), TemperatureSensor (ДатчикТемпературы).

*Атрибут -* это именованное свойство класса, включающее описание множества значений, которые могут принимать экземпляры этого свойства. Класс может иметь любое число атрибутов или не иметь их вовсе. Атрибут представляет некоторое свойство моделируемой сущности, общее для всех объектов данного класса. Например, у любой стены есть высота, ширина и толщина; при моделировании клиентов можно задавать фамилию, адрес, но­мер телефона и дату рождения. Таким образом, атрибут является абстракци­ей данных объекта или его состояния. В каждый момент времени любой ат­рибут объекта, принадлежащего данному классу, обладает вполне опреде­ленным значением. Атрибуты представлены в разделе, который расположен под именем класса; при этом указываются только их имена (рис. Л4.1).

Имя атрибута, как и имя класса, может быть произвольной текстовой строкой. На практике для именования атрибута используют одно или не­сколько коротких существительных, соответствующих некоторому свойству объемлющего класса. Каждое слово в имени атрибута, кроме самого первого, обычно пишется с заглавной буквы, например name или loadBearing. При описании атрибута можно явным образом указывать его класс и начальное значение, принимаемое по умолчанию

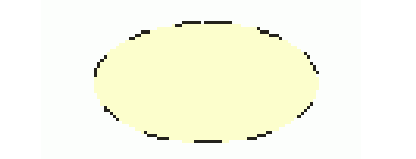
*Операцией* называется реализация услуги, которую можно запросить у любого объекта класса для воздействия на поведение. Иными словами, опе­рация - это абстракция того, что позволено делать с объектом. У всех объек­тов класса имеется общий набор операций. Класс может содержать любое число операций или не содержать их вовсе. Например, для всех объектов класса Rectangle (Прямоугольник) из библиотеки для работы с окнами, со­держащейся в пакете awt языка Java, определены операции перемещения, из­менения размера и опроса значений свойств. Часто (хотя не всегда) обраще­ние к операции объекта изменяет его состояние или его данные. Операции класса изображаются в разделе, расположенном ниже раздела с атрибутами. Имя операции, как и имя класса, может быть произвольной текстовой стро­кой. На практике для именования операций используют короткий глагол или глагольный оборот, соответствующий определенному поведению объемлю-щего класса. Каждое слово в имени операции, кроме самого первого, обычно пишут с заглавной буквы, например move или isEmpty.

Если строится дом, то такие сущности, как стены, окна, двери, встро­енные шкафы, потолки и осветительные приборы, станут частью словаря. Но ни одна из них не существует сама по себе. Стены соединены с другими сте­нами. Окна и двери располагаются в стенах, обеспечивая проходы для людей и пропуская свет. Встроенные шкафы и осветительные приборы физически связаны со стенами и потолком. Группируя все эти объекты, создаются сущ­ности более высокого уровня, например комнаты.

*Интерфейс* (Interface) - это совокупность операций, которые опреде­ляют сервис (набор услуг), предоставляемый классом или компонентом. Та­ким образом, интерфейс описывает видимое извне поведение элемента. Ин­терфейс может представлять поведение класса или компонента полностью или частично; он определяет только спецификации операций (сигнатуры), но никогда - их реализации. Графически интерфейс изображается в виде круга, под которым пишется его имя, как показано на рисунке. Интерфейс редко существует сам по себе - обычно он присоединяется к реализующему его классу или компоненту.

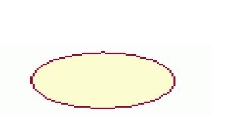


*Кооперация* (Collaboration) определяет взаимодействие; она представ­ляет собой совокупность ролей и других элементов, которые, работая совме­стно, производят некоторый кооперативный эффект, не сводящийся к про­стой сумме слагаемых. Кооперация, следовательно, имеет как структурный, так и поведенческий аспект. Один и тот же класс может принимать участие в нескольких кооперациях; таким образом, они являются реализацией образцов поведения, формирующих систему. Графически кооперация изображается в виде эллипса, ограниченного пунктирной линией, в который обычно заклю­чено только имя, как показано на рисунке.



Start Race

*Прецедент* (Use case) - это описание последовательности выполняемых системой действий, которая производит наблюдаемый результат, значимый для какого-то определенного *актера* (Actor). Прецедент применяется для структурирования поведенческих сущностей модели. Прецеденты реализу­ются посредством кооперации. Графически прецедент изображается в виде ограниченного непрерывной линией эллипса, обычно содержащего только его имя, как показано на рисунке.

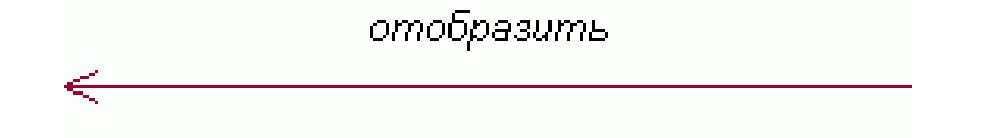


*Компонент* (Component) - это физическая заменяемая часть системы, которая соответствует некоторому набору интерфейсов и обеспечивает его реализацию. В системе можно встретить различные виды устанавливаемых компонентов, такие как СОМ+ или Java Beans, а также компоненты, являю­щиеся артефактами процесса разработки, например файлы исходного кода. Компонент, как правило, представляет собой физическую упаковку логиче­ских элементов, таких как классы, интерфейсы и кооперации. Графически компонент изображается в виде прямоугольника с вкладками, содержащего обычно только имя, как показано на рисунке. Компонент подобен классу: он описывает совокупность объектов с общими атрибутами, операциями, отно­шениями и семантикой.

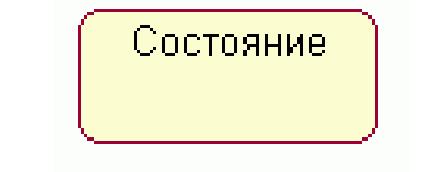


***Поведенческие сущности*** (Behavioral things) являются динамическими составляющими модели UML. Это глаголы языка: они описывают поведение модели во времени и пространстве. Существует всего два основных типа по­веденческих сущностей: *взаимодействие* и *автомат.*

*Взаимодействие* (Interaction) - это поведение, суть которого заключа­ется в обмене сообщениями (Messages) между объектами в рамках конкрет­ного контекста для достижения определенной цели. С помощью взаимодей­ствия можно описать как отдельную операцию, так и поведение совокупно­сти объектов. Взаимодействие предполагает ряд других элементов, таких как сообщения, последовательности действий (поведение, инициированное со­общением) и связи (между объектами). Графически сообщения изображают­ся в виде стрелки, над которой почти всегда пишется имя соответствующей операции, как показано на рисунке.



*Автомат* (State machine) - это алгоритм поведения, определяющий по­следовательность состояний, через которые объект или взаимодействие про­ходят на протяжении своего жизненного цикла в ответ на различные собы­тия, а также реакции на эти события. С помощью автомата можно описать поведение отдельного класса или кооперации классов. С автоматом связан ряд других элементов: *состояния, переходы* (из одного состояния в другое), *события* (сущности, инициирующие переходы) и *виды действий* (реакция на переход). Графически состояние изображается в виде прямоугольника с за­кругленными углами, содержащего имя и, возможно, подсостояния.

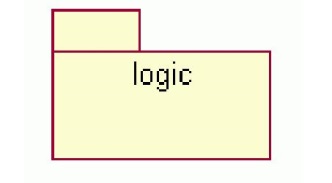


Эти два элемента - *взаимодействия* и *автоматы -* являются основны­ми поведенческими сущностями, входящими в модель UML. Семантически они часто бывают связаны с различными структурными элементами, в пер­вую очередь - *классами, кооперациями* и *объектами.*

***Группирующие сущности*** являются организующими частями модели UML. Это блоки, на которые можно разложить модель. Есть только одна первичная группирующая сущность, а именно *пакет.*

*Пакеты* (Packages) представляют собой универсальный механизм ор­ганизации элементов в группы. В пакет можно поместить структурные, пове­денческие и даже другие группирующие сущности. В отличие от компонен­тов, существующих во время работы программы, пакеты носят чисто концеп­туальный характер, то есть существуют только во время разработки.

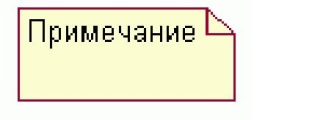
Изображается пакет в виде папки с закладкой, содержащей, как прави­ло, только имя и иногда - содержимое.



*Пакеты -* это основные группирующие сущности, с помощью которых можно организовать модель UML. Существуют также вариации пакетов, на­пример *каркасы* (Frameworks), *модели* и *подсистемы.*

***Аннотационные сущности*** *-* пояснительные части модели UML. Это комментарии для дополнительного описания, разъяснения или замечания к любому элементу модели. Имеется только один базовый тип аннотационных элементов - *примечание.*

*Примечание* (Note) - это просто символ для изображения комментариев или ограничений, присоединенных к элементу или группе элементов. Графи­чески примечание изображается в виде прямоугольника с загнутым краем, содержащим текстовый или графический комментарий, как показано на ри­сунке.



Этот элемент является основной аннотационной сущностью, которую можно включать в модель UML. Чаще всего примечания используются, что­бы снабдить диаграммы комментариями или ограничениями, которые можно выразить в виде неформального или формального текста. Существуют ва­риации этого элемента, например требования, где описывают некое жела­тельное поведение с точки зрения внешней по отношению к модели.

Между сущностями можно обнаружить не только структурные отно­шения. В вашем доме наверняка есть окна, но, вполне возможно, разного ти­па. Это могут быть большие не открывающиеся "окна-фонари" или маленькие кухонные окошки. Некоторые открываются вверх и вниз, другие раздвигают­ся в стороны. Рама может быть одинарная или двойная. Независимо от пере­численных различий, всем этим видам присуще некоторое общее свойство "оконности": каждое окно встроено в стенной проем, чтобы пропускать свет, воздух, а иногда и людей.

Л4.3.2 Отношения

На языке UML способы, которыми элементы связаны друг с другом, моделируются в виде *отношений* (*Relationship*)*.* В языке UML определены четыре типа от­ношений: *зависимость* (*Dependency*)*, ассоциация* (*Association*)*, обобщение* (*Generalization*) *и реализация* (*Realization*)*.* Графически отношение представляется линией, тип которой зависит от вида отношения.

Для каждого из названных типов отношений язык UML предоставляет графическое изображение, как показано на рис. Л4.2. Эта нотация позволяет ви­зуализировать моделируемые отношения независимо от применяемого языка программирования, так чтобы подчеркнуть их наиболее важные составляю­щие: *имя, связываемые сущности* и *свойства.*

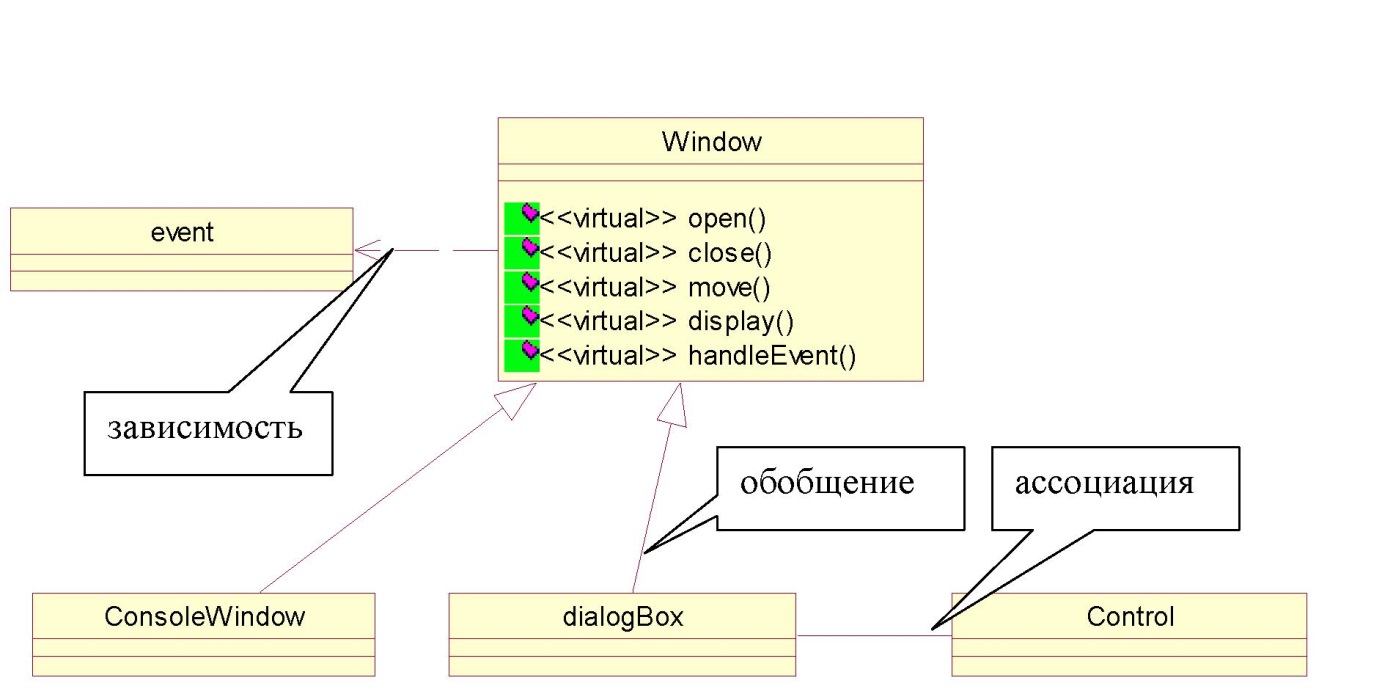


Рис. Л4.2 – Отношения

Отношения *зависимости -* это отношения использования. Например, трубы отопления зависят от нагревателя, подогревающего воду, которая в них течет. Чаще всего зависимости применяются при работе с классами, чтобы отразить тот факт, что один класс использует другой в качестве аргумента.

Графически зависимость изображается в виде прямой пунктирной линии, часто со стрелкой, которая может содержать метку, направленной от данного элемента на тот, от которого он зависит.



Отношения *ассоциации -* структурные взаимосвязи между объектами: комнаты состоят из стен, в которые могут быть встроены двери и окна; ино­гда через стены проходят трубы отопления. *Ассоциацией* называется структурное отношение, показы­вающее, что объекты одного типа неким образом связаны с объектами друго­го типа. Если между двумя классами определена ассоциация, то можно пере­мещаться от объектов одного класса к объектам другого. Вполне допустимы случаи, когда оба конца ассоциации относятся к одному и тому же классу. Это означает, что с объектом некоторого класса позволительно связать дру­гие объекты из того же класса. Ассоциация, связывающая два класса, называ­ется бинарной. Можно, хотя это редко бывает необходимым, создавать ассо­циации, связывающие сразу несколько классов; они называются n-арными.

Графически ассоциация изображается в виде линии, соединяющей класс сам с собой или с другими классами:



Ассоциации может быть присвоено имя, описывающее природу отношения. Чтобы избежать возможных двусмысленностей в понимании имени, достаточно с помощью черного треугольника указать направление, в котором оно должно читаться. Обычно имя ассоциации не указывается, если только необходимо явно задать для нее ролевые имена или в модели на­столько много ассоциаций, что возникает необходимость ссылаться на них и отличать друг от друга. Имя будет особенно полезным, если между одними и теми же классами существует несколько различных ассоциаций.

Отношения *обобщения* связывают общие классы со специализирован­ными. Например, "фонарь" - это окно с большой нераздвижной ра­мой, а "патио" - окно, раздвигающееся в стороны. *Обобщение* - это отношение между общей сущностью (суперклассом, или родителем) и ее конкретным воплощением (субклассом, или потомком). Обобщение означает, что объекты класса-потомка могут использоваться всюду, где встречаются объекты класса-родителя, но не наоборот. При этом потомок наследует свой­ства родителя, в частности его атрибуты и операции. Часто, хотя и не всегда, у потомков есть и свои собственные атрибуты и операции, помимо тех, что существуют у родителя. Операция потомка с той же сигнатурой, что и у ро­дителя, замещает операцию родителя; это свойство называют *полиморфиз­мом* (Polymorphism). Класс может иметь одного или нескольких родителей или не иметь их вовсе. Класс, у которого нет родителей, но есть потомки, называется базовым (base) или корневым (root), а тот, у которого нет потомков, - листовым (leaf). О классе, у которого есть только один родитель, говорят, что он использует одиночное наследование (Single inheritance); если родителей несколько, речь идет о множественном наследовании (Multiple inheritance). Обобщение чаще всего используют между классами и интерфейсами, чтобы показать отношения наследования. Графически отношение обобщения изображается в виде линии с большой незакрашенной стрелкой, направленной на родителя (рис. Л4.3).

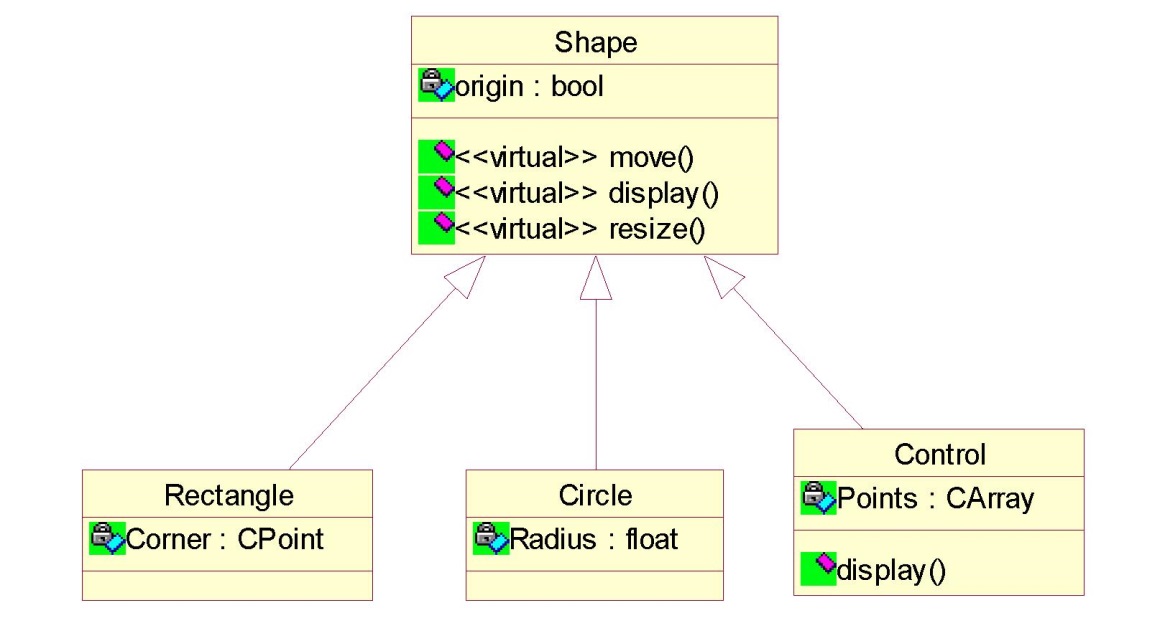


Рис. Л4.3 – Обобщение

Эти три типа отношений охватывают большую часть способов взаимо­действия элементов.

Наконец, *реализация*  - это семантическое отношение меж­ду классификаторами, при котором один классификатор определяет "кон­тракт", а другой гарантирует его выполнение. Отношения реализации встре­чаются в двух случаях: 1) между интерфейсами и реализующими их классами или компонентами, 2) между прецедентами и реализую­щими их кооперациями. Отношение реализации изображается в виде пунк­тирной линии с незакрашенной стрелкой, как нечто среднее между отноше­ниями *обобщения* и *зависимости.*



Четыре описанных элемента являются основными типами отношений, которые можно включать в модели UML. Существуют также их вариации, например ***уточнение*****(Refinement), *трассировка* (Trace), *включение* и *расши­рение*** (для зависимостей).

Л4.3.3 Диаграммы

*Диаграмма* в UML - это графическое представление набора элементов, изображаемое чаще всего в виде связанного графа с вершинами (сущностя­ми) и ребрами (отношениями). Диаграммы рисуют для визуализации систе­мы с разных точек зрения. Диаграмма - в некотором смысле одна из проек­ций системы. Как правило, за исключением наиболее тривиальных случаев, диаграммы дают свернутое представление элементов, из которых составлена система. Один и тот же элемент может присутствовать во всех диаграммах, или только в нескольких (самый распространенный вариант), или не присут­ствовать ни в одной (очень редко). Теоретически диаграммы могут содержать любые комбинации сущностей и отношений. На практике, однако, применя­ется сравнительно небольшое количество типовых комбинаций, соответст­вующих пяти наиболее употребительным видам, которые составляют архи­тектуру проектируемой системы.

Обычно в UML выделяют группу в составе 8-9 типовых или канонических диаграмм, на основе которых могут формироваться другие более узко специализированные диаграммы. Как правило, к числу канонических относят следующие типы диаграмм (diagram): классов (class), объектов (object), прецедентов (use case), последовательностей (sequence), кооперации (collaboration), состояний (statechart), действий (activity), компонентов (component), развертывания (deployment).

На *диаграмме классов* показывают классы, интерфейсы, объекты и кооперации, а также их отношения. При моделировании объектно-ориентированных систем этот тип диаграмм используют чаще всего. Диа­граммы классов соответствуют статическому виду системы с точки зрения проектирования. Диаграммы классов, которые включают активные классы, соответствуют статическому виду системы с точки зрения процессов. Пример диаграммы классов представлен на рисунке Л4.4.

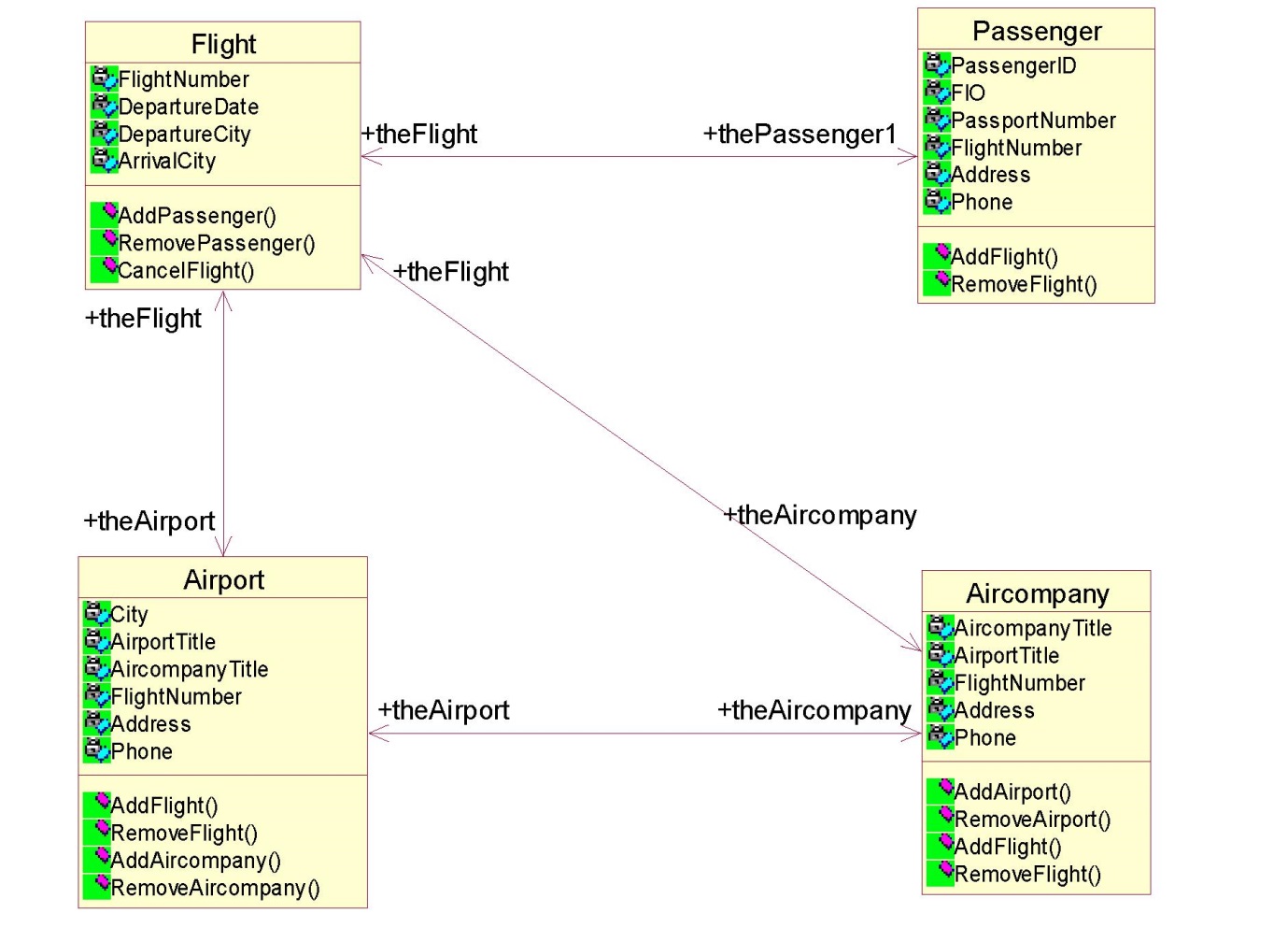


Рис. Л4.4 – Пример диаграммы классов

На *диаграмме объектов* представлены объекты и отношения между ними. Они являются статическими "фотографиями" экземпляров сущностей, показанных на диаграммах классов. Диаграммы объектов, как и диаграммы классов, относятся к статическому виду системы с точки зрения проектиро­вания или процессов, но с расчетом на настоящую или макетную реализа­цию.

На *диаграмме прецедентов* представлены прецеденты и актеры (част­ный случай классов), а также отношения между ними. Диаграммы прецеден­тов относятся к статическому виду системы с точки зрения прецедентов ис­пользования. Они особенно важны при организации и моделировании пове­дения системы.

*Диаграммы последовательностей* и *кооперации* являются частными случаями *диаграмм взаимодействия.* На *диаграммах взаимодействия* пред­ставлены связи между объектами; показаны, в частности, сообщения, кото­рыми объекты могут обмениваться. *Диаграммы взаимодействия* относятся к динамическому виду системы. При этом *диаграммы последовательности* от­ражают временную упорядоченность сообщений, а *диаграммы кооперации -*структурную организацию обменивающихся сообщениями объектов. Эти диаграммы изоморфны, то есть могут быть преобразованы друг в друга.

На *диаграммах состояний* представлен автомат, включающий в себя состояния, переходы, события и виды действий. Диа­граммы состояний относятся к динамическому виду системы; особенно они важны при моделировании поведения интерфейса, класса или кооперации. Они акцентируют внимание на поведении объекта, зависящем от последова­тельности событий, что очень полезно для моделирования реактивных сис­тем.

*Диаграмма деятельности -* это частный случай диаграммы состояний; на ней представлены переходы потока управления от одной деятельности к другой внутри системы. *Диаграммы деятельности* относятся к динамиче­скому виду системы; они наиболее важны при моделировании ее функцио­нирования и отражают поток управления между объектами.

На *диаграмме компонентов* представлена организация совокупности компонентов и существующие между ними зависимости. *Диаграммы компо- нентов* относятся к статическому виду системы с точки зрения реализации. Они могут быть соотнесены с *диаграммами классов,* так как компонент обычно отображается на один или несколько классов, интерфейсов или коо­пераций.

На *диаграмме развертывания* представлена конфигурация обрабаты­вающих узлов системы и размещенных в них компонентов. *Диаграммы раз­вертывания относятся* к статическому виду архитектуры системы с точки зрения развертывания. Они связаны с *диаграммами компонентов,* поскольку в узле обычно размещаются один или несколько компонентов.

Здесь приведен неполный список диаграмм, применяемых в UML. Ин­струментальные средства позволяют генерировать и другие диаграммы, но девять перечисленных встречаются на практике чаще всего.

Л4.3.4 Правила и общие механизмы языка UML

Строительные блоки UML нельзя произвольно объединять друг с дру­гом. Как и любой другой язык, UML характеризуется набором правил, опре­деляющих, как должна выглядеть хорошо оформленная модель, то есть се­мантически самосогласованная и находящаяся в гармонии со всеми моделя­ми, которые с нею связаны.

В языке UML имеются *семантические правила* (см. раздел 5 конспекта лекций по курсу ПАС), позволяющие коррект­но и однозначно определять:

*имена,* которые можно давать сущностям, отношениям и диаграм­мам;

*область действия* (контекст, в котором имя имеет некоторое значе­ние);

*видимость* (когда имена видимы и могут использоваться другими элементами);

*целостность* (как элементы должны правильно и согласованно со­относиться друг с другом);

*выполнение* (что значит выполнить или имитировать некоторую ди­намическую модель).

Модели, создаваемые в процессе разработки программных систем, эво­люционируют со временем и могут неоднозначно рассматриваться разными участниками проекта в разное время. По этой причине создаются не только хорошо оформленные модели, но и такие, которые:

* неполные (отдельные элементы пропущены или скрыты, чтобы упростить восприятие);

- несогласованные (целостность модели не гарантируется).

Появление не слишком хорошо оформленных моделей неизбежно в процессе разработки, пока не все детали системы прояснились в полной ме­ре. Правила языка UML побуждают в ходе работы над моделью решать наиболее важные вопросы анализа, проектирования и реа­лизации, в результате чего модель со временем становится хорошо оформ­ленной.

Любое строительство упрощается и ведется более эффективно, если придер­живаться некоторых соглашений. Следуя определенным архитектурным об­разцам, можно оформить здание в викторианском или французском стиле. Тот же принцип применим и в отношении UML. Последовательно используемые на практике пе­речисленные ниже *общие механизмы* суще­ственно облегчают работу с этим языком:

* спецификации (Specifications);
* дополнения (Adomments);
* принятые деления (Common divisions);
* механизмы расширения (Extensibility mechanisms).

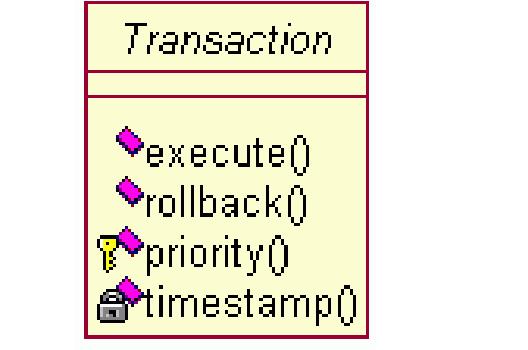
*Спецификация*

UML - это не просто графический язык. За каждой частью его системы графической нотации стоит *спецификация,* содержащая текстовое представ­ление синтаксиса и семантики соответствующего строительного блока. На­пример, пиктограмме класса соответствует спецификация, полностью описы­вающая его атрибуты, операции (включая полные сигнатуры) и поведение, хотя визуально пиктограмма порой отражает только малую часть этой сово­купности. Более того, может существовать другое представление этого клас­са, отражающее совершенно иные его аспекты, но, тем не менее, соответст-вующее все той же спецификации. С помощью графической нотации UML вы визуализируете систему, с помощью спецификаций UML - описываете ее детали. Таким образом, допустимо строить модель инкрементно, то есть по­шаговым образом - сначала нарисовать диаграмму, а потом добавить семан­тику в спецификацию модели, или наоборот - начать со спецификации (воз­можно, применив обратное проектирование к существующей системе), а по­том на ее основе создавать диаграммы.

Спецификации UML создают семантический задний план, который полностью включает в себя составные части всех моделей системы, согласованные между собой. Таким образом, диаграммы UML можно считать визуальными проекциями на этот задний план, при этом каждая из них раскрывает один из значимых аспектов системы.

Почти каждый из элементов UML имеет соответствующее ему уни­кальное графическое обозначение, которое дает визуальное представление о самых важных аспектах этого элемента. Например, обозначение класса спе­циально придумано так, чтобы его было легко рисовать, поскольку класс -наиболее употребительный элемент при моделировании объектно-ориентированных систем. Нотация класса содержит самые важные его харак­теристики: имя, атрибуты и операции.

Спецификация класса может содержать и другие детали, например ви­димость атрибутов и операций или указание на то, что класс является абст­рактным. Многие такие детали можно визуализировать в виде графических или текстовых дополнений к стандартному прямоугольнику, служащему изо­бражением класса. Так, на рисунке показан класс, в обозначение которого включены сведения о том, что он абстрактный и содержит две открытые, од­ну защищенную и одну закрытую операцию.



*Дополнения*

Каждый элемент нотации UML содержит базовый для него символ, к которому можно добавлять разнообразные специфичные для него дополне­ния.

*Принятые деления*

При моделировании объектно-ориентированных систем реальность разделяется с учетом, по крайней мере, двух подходов.

Прежде всего, существует разделение на классы и объекты. Класс - это абстракция, объект - конкретная материализация этой абстракции. В языке UML можно моделировать и классы, и объекты.

Практически все строительные блоки UML характеризуются дихото­мией "класс/объект". Так, имеются прецеденты и экземпляры прецедентов, компоненты и экземпляры компонентов, узлы и экземпляры узлов и т.д.

В графическом представлении для объекта принято использовать тот же сим­вол, что и для его класса, а название объекта подчеркивать.

Еще одним вариантом разделения является деление на интерфейс и его реализацию. Интерфейс декларирует контракт, а реализация представляет конкретное воплощение этого контракта и обязуется точно следовать объяв­ленной семантике интерфейса. UML позволяет моделировать обе эти катего­рии, интерфейсы и их реализации. Почти все строительные блоки UML ха­рактеризуются дихотомией "интерфейс/реализация". Например, прецеденты реализуются кооперациями, а операции - методами.

*Механизмы расширения*

UML - это стандартный язык для разработки "чертежей" систем и их программного обеспечения, но ни один замкнутый язык не в состоянии охватить нюансы всех возможных моделей в различных предметных областях. Поэтому UML является открытым языком, то есть допускает контролируемые расширения. Механизмы расширения UML включают: стереотипы; помеченные значения; ограничения.

*Стереотип* (Stereotype) расширяет словарь UML, позволяя на основе существующих блоков языка создавать новые, специфичные для решения конкретной проблемы. Например, работая с такими языками программирова­ния, как Java или C++, часто приходится моделировать исключения (Exceptions) - они являются обыкновенными классами, хотя и рассматрива­ются особым образом. Обычно требуется, чтобы исключения можно было возбуждать и перехватывать, и ничего больше. Если пометить исключения соответствующим стереотипом, то с ними можно будет обращаться как с обычными строительными блоками языка.

*Помеченное значение* (Tagged value) расширяет свойства строительных блоков UML, позволяя включать новую информацию в спецификацию эле­мента. Скажем, если вы работаете над "коробочным" продуктом и выпускае­те много его версий, то зачастую необходимо отслеживать версию и автора какой-нибудь важной абстракции. Ни версия, ни автор не являются первич­ными концепциями UML, но их можно добавить к любому блоку, такому, например, как класс, задавая для него новые помеченные значения.

*Ограничения* (Constraints) расширяют семантику строительных блоков UML, позволяя определять новые или изменять существующие правила.

Совместно эти три механизма расширения языка позволяют модифи­цировать UML в соответствии с потребностями проекта. С помощью механизмов расши­рения можно создавать новые строительные блоки, модифицировать сущест­вующие и даже изменять их семантику.

## Л4.4 Общее описание характеристик и возможностей CASE-средства проектирования Sparx Enterprise Architect

В данной лабораторной работе предлагается к изучению и использованию одно из наиболее распространенных и удобных CASE-средств поддержки процесса проектирования Enterprise Architect, разработанное американской фирмой Sparх [5]. Sparх Enterprise Architect (SEA) является высокопроизводительным инструментом, основанным на стандарте UML 2.1, который используется для моделирования и создания автоматизированных систем и их программного обеспечения. Покрывает весь процесс разработки от формирования требований к системе до её полной реализации. Предоставляет средства надежной и эффективной визуализации и организации взаимодействия в коллективе. Обеспечивает поддержку тестирования, управления сопровождением и изменениями. Характеризуется низкими издержками на установку, высокой производительностью и интуитивно понятным интерфейсом.

Обладает следующими особенностями:

* поддерживаются все диаграммы UML;
* поддерживает генерацию и обратный инжиниринг исходного кода для многих популярных языков, включая: C++, C#, Java, Delphi, VB.Net, Visual Basic, ActionScript, PHP, Python и др.;
* помимо работы с исходным кодом, поддерживает обратный инжиниринг файлов Java .jar и бинарных сборок .Net;
* имеет мощные инструменты для создания документации и разного рода отчетов (имеет WYSIWYG редактор шаблонов);
* позволяет моделировать бизнес-процессы, веб-сайты, пользовательские интерфейсы, сети, конфигурации аппаратного обеспечения, сообщения и т.д., оценивать размер трудозатрат проектных работ в часах, фиксировать и трассировать требования, ресурсы, тест-планы, дефекты и запросы на изменения;
* позволяет проектировать и генерировать элементы баз данных (БД). В качестве сервера БД могут использоваться SQL Server, MySQL, Oracle 9i и 10g, PostgreSQL, Adaptive Server Anywhere, MSDE Server, Progress OpenEdge.

Архитектурно Enterprise Architect представляет собой программу – рабочее место SEA, из которого осуществляется соединение через собственный драйвер БД с проектным репозиторием, организованным в виде базы данных.

На рабочем месте хранятся пользовательские настройки этого рабочего места: настройки отображения панелей инструментов, набор горячих клавиш и т.д.

В проектном репозитории хранятся следующие элементы моделирования:

* объекты модели, такие как UML-элементы и пакеты;
* коннекторы, которые связывают взаимодействующие объекты;
* диаграммы, отображающие объекты, коннекторы и ссылки на другие диаграммы.

При этом один элемент может быть отображен на нескольких диаграммах, но физически как объект базы данных он хранится только в одном экземпляре. В связи с этим удаление элемента на диаграмме не вызывает удаление объекта из репозитория.

Также в проектном репозитории хранится дополнительная и служебная информация. Для обмена информацией между репозиториями используется экспорт/импорт файлов XML-формата.

## Л4.5 Описание интерфейса и основ работы с системой Sparx Enterprise Architect

*Л4.5.1* Рабочее пространство SEA

Рабочее пространство (Workspace) SEA состоит из многих окон. Вместе, эти элементы предоставляют простую и удобную среду разработки. В общем, рабочее пространство SEA похоже на такие программы как, например Microsoft Outlook или Microsoft Visual Studio.

Компоненты рабочего пространства SEA:

* Главное меню и панели инструментов (Main Menu and Toolbars) – находятся сверху рабочего пространства. Главное меню предоставляет доступ к подменю. Панели инструментов могут добавляться и убираться по мере необходимости.
* Контекстные меню (Context Menus) – повсюду в SEA реализована система контекстных меню, возникающих по нажатию правой кнопки «мыши».
* Комбинации клавиш (Key Combinations) – функции главного меню и контекстных меню имеют горячие комбинации клавиш. Для просмотра горячих клавиш можно использовать внутреннюю справочную систему SEA. Для этого необходимо выбрать раздел главного меню Помощь (Help), и выбрать справку по горячим клавишам (Keyboard Accelerator Map). Также все горячие клавиши можно перенастроить под себя в разделе меню Customize.
* Панель инструментов UML (Architect Enterprise UML Toolbox) – панель инструментов в стиле Outlook, из которой можно выбирать элементы и связи модели для добавления на диаграммы. Это очень важная функция SEA, так как она предоставляет все компоненты и связи, которые можно использовать в построении диаграмм.
* Просмотрщик диаграмм (Diagram View) – здесь можно добавлять к модели новые элементы, а также устанавливать их характеристики на диаграмме.
* Проводник проекта (Project Browser) – используется для навигации по проекту. Чтобы просмотреть проект, необходимо кликнуть на иконку “плюс”, чтобы раскрыть папку или пакет и затем совершить двойной щелчок по иконке диаграммы, появившейся под названием пакета (папки). Выбранная диаграмма отображается в окне просмотра диаграмм (Diagram View) в центре экрана.
* Окно быстрой справки (Task Pane) – предоставляет быстрый доступ к справочной информации по SEA, онлайн ресурсам Sparx Systems.
* Внешний вид (Visual Style) – вы можете настроить внешний вид приложения от обычного Windows-стиля до современного Vista-стиля.

Л4.5.2 Стартовая страница SEA

После запуска Enterprise Architect, перед вами возникает стартовая страница (Start Page). С помощью этой страницы вы можете вызвать следующие опции системы:

* Поиск (Search) – для того, чтобы найти какой-либо объект в SEA , необходимо ввести название объекта в это поле и нажать на кнопку […] . SEA отобразит результаты поиска в окне Поиск Модели (Model Search). Кликнув на результат поиска, можно выделить его в Проводнике Проекта (Project Browser).
* Настройки системы (Configure Options) – выводит диалог настроек (Options Dialog), при помощи которого можно настроить то, каким образом будет отображать и обрабатывать информацию.
* Интернет ресурсы и учебники (Online Resources & Tutorials) – открывает страницу сайта Sparx Systems, содержащую множество интернет ресурсов, учебников, примеров, презентаций и т.д., посвященных SEA и UML.
* Открыть файл проекта (Open a Project File) – выводит диалог открытия файла проекта.
* Создать новый проект (Create a New Project) – создает новый проект и выводит мастер модели (Model Wizard).
* Скопировать базовый проект (Copy a Base Project) – создает новую модель на основе выбранного базового проекта.
* Подключиться к серверу (Connect to Server) – позволяет определить имя Хранилища Данных (Data Source) для подключения к любому из допустимых серверов БД (функция доступна только в корпоративной версии программы).
* Последние (Recent) - содержит список наиболее часто используемых проектов.

Л4.5.3 Главное меню SEA

Главное меню (Main Menu) предоставляет доступ ко многим высокоуровневым функциям, относящимся к жизненному циклу проекта и к администрированию. Рассмотрим все меню по порядку:

* Меню ‘Файл’ (File) предоставляет возможности создавать, открывать, закрывать и сохранять проекты, а также выполнять операции вывода на принтер (см. Приложение А, таблица А.1).
* Меню ‘Редактировать’ (Edit) предоставляет широкий спектр функций, применимых для элементов открытой диаграммы (см. Приложение А, таблица А.2).
* Меню ‘Просмотр’ (View) предоставляет возможность настроить внешний вид системы (см. Приложение А, таблица А.4).
* Меню ‘Проект’ (Project) предоставляет доступ к управлению проектом (см. Приложение А, таблица А.6)
* Меню ‘Диаграмма’ (Diagram) позволяет сохранять изображения диаграмм в файл, а так же конфигурировать свойства и опции диаграмм (см. Приложение А, таблица А.10).
* Меню ‘Элемент’ (Element) позволяет конфигурировать атрибуты элементов, контролировать их местоположение, генерировать документацию и управлять ресурсами проекта (см. Приложение А, таблица А.11).
* Меню ‘Инструменты’ (Tools) предоставляет доступ к множеству различных инструментов, включая относящиеся к кодогенерации, управлению файлами .EAP, внешним ресурсам и т.д. (см. Приложение А, таблица А.14).
* Меню ‘Настройки’ (Settings) предоставляет доступ к различным настройкам всего проекта (см. Приложение А, таблица А.15).
* Меню ‘Окно’ (Window) предоставляет доступ к различным действиям с открытыми окнами.
* Меню ‘Помощь’ (Help) предоставляет доступ к файлам помощи SEA, к файлам ReadMe, к лицензионному соглашению SEA и различным специальным функциям официального сайта Sparx Systems.

Л4.5.4 Создание нового проекта

**4.5.4.1** Первая задача, которую можно выполнить в SEA– это создание нового проекта (project) и добавление в него пакетов, диаграмм, элементов и коннекторов.

После запуска Enterprise Architect открывается стартовое окно системы (Start Page). Для создания нового проекта:

1. Нажмите на Create a New Project на стартовой странице системы (либо File->New Project). Появится диалог создания нового проекта (New Project).
2. Выберите местоположение нового проекта, его имя и нажмите на кнопку Сохранить (Save). Появиться окно мастера модели (Model Wizard).
3. Выбрать один или более шаблонов модели (model templates), расставив галочки, где необходимо.
4. Нажать кнопку Ok. SEA создаст ваш проект и отобразит его в Проводнике Проекта (Project Browser), в правой части окна.

Чтобы просмотреть проект, необходимо щелкнуть кнопкой «мыши» (кликнуть) по иконке “плюс”, чтобы раскрыть папку или пакет и затем совершить двойной щелчок по иконке диаграммы, появившейся под названием пакета (папки). Выбранный шаблон диаграммы отображается в окне просмотра диаграмм (Diagram View) в центре экрана.

**4.5.4.2** Пакет (Package) – контейнер, содержащий элементы модели, отображается в окне проводника проекта (Project Browser). Содержание пакета сортируется в алфавитном порядке.

Чтобы добавить новый пакет, необходимо либо нажать на соответствующую иконку на панели инструментов проводника, либо воспользоваться контекстным меню (Add->Add Package), либо использовать горячие клавиши (CTRL+W), далее ввести необходимое имя пакета и нажать на кнопку OK.

**4.5.4.3** Диаграмма – это представление компонентов или элементов модели и, в зависимости от типа модели, того, как эти элементы связаны или как они взаимодействуют друг с другом.

Чтобы добавить новую диаграмму, необходимо либо нажать на соответствующую иконку на панели инструментов проводника, либо воспользоваться контекстным меню (Add->Add Diagram). Появиться диалоговое окно добавления новой диаграммы. В этом окне необходимо выбрать категорию диаграммы в поле Select From и тип диаграммы в форме Diagram Types. Новая диаграмма появится в выбранном пакете. Для нее в центре экрана откроется просмотрщик диаграмм (Diagram View).

**4.5.4.4** Существует несколько способов добавления элементов в диаграмму и/или пакет. Самый простой метод – использовать панель инструментов UML (Enterprise Architect UML Toolbox) слева от диаграммы, который автоматически предоставляет список элементов, применимых для выбранного типа диаграммы. Необходимо перетащить необходимый элемент из этого списка на диаграмму. Появляется диалог свойств элемента (в случае, если он не появляется, необходимо дважды кликнуть по добавленному элементу). После введения всех необходимых свойств и параметров элемента он появляется в проводнике проекта в соответствующем пакете. Для получения дополнительной справки о добавленном элементе можно выбрать пункт UML Help из контекстного меню этого элемента.

Подробнее некоторые базовые элементы будут написаны ниже.

**4.5.4.5** Связи или коннекторы (Connectors) определяют специальные отношения между элементами. Для создания связи на диаграмме необходимо выбрать нужный тип коннектора из панели инструментов UML (Enterprise Architect UML Toolbox), которая находиться слева от диаграммы, нажать на нужный элемент на диаграмме и перетащить указатель ко второму элементу. Если дважды кликнуть по появившемуся коннектору, появиться диалог свойств связи, в котором можно определить нужные характеристики связи.

**4.5.4.6** Для перемещения элементов в окне проводника проекта, используйте кнопки перемещения на панели инструментов проводника, либо в контекстном меню. Также можно перемещать элементы простым перетягиванием с помощью курсора мышки. Аналогично можно перемещать элементы в окне просмотрщика диаграмм.

Для удаления элементов из окна проводника проекта или из окна просмотрщика диаграмм, необходимо выбрать операцию Delete из контекстного меню.

Для сохранения диаграммы можно использовать горячие клавиши (CTRL+S) либо воспользоваться панелью главного меню (Diagram->Save).

Л4.5.5 Шаблоны моделей в SEA

Содержащиеся в SEA шаблоны моделей разработаны для того, чтобы помочь в создании проектов и моделей, как для опытных пользователей, так и для новичков. Каждый шаблон представляет собой структуру, на основе которой можно создать необходимую модель. Рассмотрим существующие шаблоны:

* Шаблон модели бизнес-процесса (Business Process Model Template) – описывает поведение и потоки информации в организации или системе. Бизнес-модель охватывает все значимые события, входные и выходные данные, ресурсы, обработку и т.п., связанные с соответствующими бизнес-процессами.
* Шаблон модели требований (requirements Model Template) – это структурированный каталог требований конечного пользователя и отношений между ними. Менеджер управления требованиями, встроенный в SEA может использоваться, чтобы определить элементы требования, соединить требования и элементы модели, объединить требования в иерархию и сообщать о требованиях.
* Шаблон модели прецедентов (Use Case Model Template) – описывает функционал системы с точки зрения прецедентов. Каждый прецедент представляет одно повторяемое действие, которое пользователь (user) или ‘актер’ (actor) может совершить, используя систему.
* Шаблон модели доменов (Domain Model Template) – высокоуровневая концептуальная модель, определяющая физические и абстрактные объекты в области проекта. Может использоваться, чтобы задокументировать связи между классами и их назначение.
* Шаблон модели классов (Class Model Template) – строгая, логическая модель программного продукта, который находиться в разработке. Классы обычно имеют прямую связь с исходным кодом или другими элементами, связанными с разработкой, которые могут объединяться в выполняемые компоненты.
* Шаблон модели баз данных (Database Model Template) – описывает данные, которые должны приниматься и храниться, как часть всей разрабатываемой системы. Т.е. модели реляционных баз данных, которые детально описывают таблицы и данные, и позволяют генерировать DDL скрипты, которые, в свою очередь, позволяют создавать и настраивать базы данных.
* Шаблон модели компонентов (Component Model Template) – определяет, как классы и остальные элементы низкого уровня собираются в высокоуровневые компоненты, а также связи между ними.
* Шаблон модели развертывания (Deployment Model Template) – описывает, как и где система должна быть развернута. Физические машины и процессоры представляются узлами, а вся внутренняя конструкция может быть отображена с использованием узлов и прочих артефактов.
* Шаблон модели тестирования (Testing Model Template) – описывает и содержит каталог тестов, планов тестирования и результаты, которые выполняются в отношении текущей модели.

## Л4.6 Описание примеров построения диаграмм

Представленные ниже примеры являются фрагментами проекта системы, организующей работу банкомата (Automated Teller Machine, ATM) по обслуживанию клиента по его кредитной карте.

Л4.6.1 Диаграмма прецедентов

**4.6.1.1** Любые системы проектируются с учетом того, что в процессе своей работы они будут использоваться людьми и/или взаимодействовать с другими системами. Сущности, с которыми взаимодействует система в процессе своей работы, называются экторами, причем каждый эктор ожидает, что система будет вести себя строго определенным, предсказуемым образом [2].

Эктор (actor, актер, действующее лицо, исполнитель) - это множество логически связанных ролей, исполняемых при взаимодействии с прецедентами или сущностями (система, подсистема или класс). Эктором может быть человек или другая система, подсистема или класс, которые представляют нечто вне сущности.

Графически эктор изображается либо "человечком", либо символом класса с соответствующим стереотипом.

Прецедент (use case, вариант использования) - описание множества последовательных событий (включая варианты), выполняемых системой, которые приводят к наблюдаемому эктором результату. Прецедент представляет поведение сущности, описывая взаимодействие между экторами и системой. Прецедент не показывает то, "как" достигается некоторый результат, а только "что" именно выполняется.

Прецеденты обозначаются очень простым образом - в виде эллипса, внутри которого указано его название.

Прецеденты и экторы соединяются с помощью линий. Часто на одном из концов линии изображают стрелку, причем направлена она к тому, у кого запрашивают сервис, другими словами, чьими услугами пользуются.

Цели создания диаграмм прецедентов [6]:

* определение границы и контекста моделируемой предметной области на ранних этапах проектирования;
* формирование общих требований к поведению проектируемой системы;
* разработка концептуальной модели системы для ее последующей детализации;
* подготовка документации для взаимодействия с заказчиками и пользователями системы.

**4.6.1.2** Прецеденты начинают описывать то, что должна будет делать ваша система. Но чтобы фактически разработать систему, потребуются более конкретные детали. Они определяются в документе, называемом "потоком событий" (flow of events). Целью потока событий является документирование процесса обработки данных, реализуемого в рамках диаграммы прецедентов. Этот документ подробно описывает, что будут делать пользователи системы и что – сама система.

Поток событий также не должен зависеть от реализации.

Обычно поток событий содержит [7]: Краткое описание, Предусловия (pre-conditions), Основной поток событий, Альтернативный поток событий, Постусловия (post-conditions).

Рассмотрим последовательно эти составные части.

Каждый прецедент должен иметь связанное с ним *краткое описание* того, что он будет делать. Например, вариант использования "Перевести деньги" системы банкомата может содержать следующее описание: вариант использования "Перевести деньги" позволяет клиенту или служащему банка переводить деньги с одного счета на другой. Описание следует делать четким и коротким, при этом оно должно определять типы пользователей, и ожидаемый ими конечный результат. Во время работы над проектом эти описания будут напоминать членам команды, почему тот или иной вариант использования был включен в проект и что он должен делать.

*Предусловия* варианта использования – это такие условия, которые должны быть выполнены, прежде чем вариант использования начнет свою работу. Например, таким условием может быть выполнение другого варианта использования или наличие у пользователя прав доступа, требуемых для запуска данного варианта использования. Не у всех вариантов использования бывают предварительные условия.

Конкретные детали вариантов использования отражаются в *основном и альтернативном потоках событий*. Поток событий поэтапно описывает, что должно происходить во время выполнения заложенной в прецеденты функциональности. Поток событий уделяет внимание тому, что (а не как) будет делать система, причем описывает это с точки зрения пользователя.

Первичный и альтернативный потоки событий содержат [7]:

* Описание того, каким образом запускается вариант использования;
* Различные пути выполнения варианта использования;
* Нормальный, или основной, поток событий варианта использования;
* Отклонения от основного потока событий (так называемые альтернативные потоки);
* Потоки ошибок;
* Описание того, каким образом завершается вариант использования.

Например, поток событий прецедента "Снять деньги" может выглядеть так, как описано ниже.

Основной поток:

1. Вариант использования начинается, когда клиент вставляет свою карточку в банкомат.
2. Банкомат выдает приветствие и предлагает клиенту ввести свой персональный идентификационный номер.
3. Клиент вводит номер.
4. Банкомат подтверждает введенный номер. Если номер не подтверждается, выполняется альтернативный поток событий А1.
5. Банкомат выводит список доступных действий:

* Положить деньги на счет
* Снять деньги со счета
* Перевести деньги

1. Клиент выбирает пункт "Снять деньги".
2. Банкомат запрашивает, сколько денег нужно снять.
3. Клиент вводит требуемую сумму.
4. Банкомат определяет, достаточно ли на счету денег. Если денег недостаточно, выполняется альтернативный поток А2. Если во время подтверждения суммы возникают ошибки, выполняется поток ошибок Е1.
5. Банкомат вычитает требуемую сумму из счета клиента.
6. Банкомат выдает клиенту требуемую сумму наличными.
7. Банкомат возвращает клиенту его карточку.
8. Вариант использования завершается.

Альтернативный поток А1 (ввод неправильного идентификационного номера):

1. ATM информирует клиента, что идентификационный номер введен неправильно.
2. ATM возвращает клиенту его карточку.
3. Вариант использования завершается.

Альтернативный поток А2 (недостаточно денег на счету):

1. ATM информирует клиента, что денег на его счету недостаточно.
2. ATM возвращает клиенту его карточку.
3. Вариант использования завершается.

Поток ошибок Е1 (ошибка в подтверждении запрашиваемой суммы):

1. ATM сообщает пользователю, что при подтверждении запрашиваемой суммы произошла ошибка, и дает ему номер телефона службы поддержки клиентов банка.
2. ATM заносит сведения об ошибке в журнал ошибок. Каждая запись содержит дату и время ошибки, имя клиента, номер его счета и код ошибки.
3. ATM возвращает клиенту его карточку.
4. Вариант использования завершается.

Документируя поток событий, можно использовать нумерованные списки (как это сделано в данном примере), ненумерованные списки, разбитый на параграфы текст и даже блок-схемы. Поток событий должен быть согласован с определенными ранее требованиями. Описывая поток, необходимо помнить о тех, кто будет читать этот документ. При его изучении заказчики будут проверять, соответствует ли он их ожиданиям, а аналитики – соответствует ли он требованиям к системе. Менеджер проекта захочет лучше понять, что же будет создано, а также сделать или обновить оценки проекта.

*Постусловиями* называются такие условия, которые должны быть выполнены после завершения варианта использования. Например, в конце варианта использования можно установить флажок. Информация такого типа входит в состав постусловий. Как и в случае предусловий, с помощью постусловий можно вводить сведения о порядке выполнения вариантов использования системы. Если, скажем, после одного из вариантов использования должен всегда выполняться другой, это можно описать как постусловие. Такие условия имеются не у каждого варианта использования.

**4.6.1.3** Существуют следующие основные виды связей для диаграмм прецедентов [4]: Use (использования); Associate (ассоциации); Generalize (обобщения); Include (включения); Extend (дополнения). Рассмотрим их.

*Связь использования* (uses relationship) позволяет одному варианту использования задействовать функциональность другого. С помощью таких связей обычно моделируют многократно применяемую функциональность, встречающуюся в двух или более вариантах использования.

*Связь ассоциации* (associate realitionship) – это связь между вариантом использования и действующим лицом. Иногда ее называют связью *коммуникации*. На языке UML связь коммуникации изображают в виде стрелки. Направление стрелки показывает, кто инициирует коммуникацию. Например, действующее лицо Клиент может инициировать коммуникацию с системой для запуска функции банкомата "Снять деньги". Вариант использования также может инициировать коммуникацию с действующим лицом.

*Связь* *обобщения* (generalization relationship) на диаграмме прецедентов используется для того, чтобы обозначить наследования экторов. Рисуется от элемента-источника к обобщенному элементу. Обобщенный элемент включает в себя характеристики источника.

*Связь* *включения* (includes relationship) используется для включения общего поведения из включаемого варианта использования в базовый вариант использования для поддержки повторного использования общего поведения. В примере ATM варианты использования "Снять деньги" и "Положить деньги на счет" должны опознать (аутентифицировать) клиента и его идентификационный номер перед тем, как разрешить выполнение самой транзакции. Вместо того чтобы подробно описывать процесс аутентификации для каждого из них, можно включить эту функциональность в свой собственный вариант использования под названием "Аутентифицировать клиента". Когда какому-нибудь варианту использования потребуется выполнить эти действия, он сможет воспользоваться функциональностью созданного варианта использования "Аутентифицировать клиента". Связь использования изображается в UML с помощью стрелок и слова «include».

*Связь* *расширения* (extends relationship) позволяет варианту использования только при необходимости применять функциональные возможности, предоставляемые другим вариантом использования. Она напоминает связь использования. В обоих типах отношений некоторая общая функциональность выделяется в отдельный вариант использования. В примере ATM вариант использования "Снять деньги" дополняется вариантом "Ускоренное снятие денег". На языке UML связи расширения изображают в виде стрелки со словом "extends" (расширение).

**4.6.1.4** Рассмотрим диаграмму вариантов использования отражающую систему работы банковского автомата (рисунок Л4.5) [8]. Для ее создания необходимо создать новый проект, добавить к нему пакет и в этом пакете добавить новую Use Case диаграмму. Экторы и прецеденты добавляются на диаграмму с помощью UML Toolbox. Так как далее будет проводиться работа с прецедентом "Снять деньги со счета", то необходимо выполнить следующее действие – в контекстном меню элемента выбрать Advanced->Composite Element.

Эта диаграмма показывает субъект банкомата (ATM), трех действующих лиц: Клиента банка, Банк и Оператора кредитной системы, шесть вариантов использования: Снять наличные, Перевести деньги со счета, Положить деньги на счет, Посмотреть баланс, Пополнить запас денег, Произвести оплату и Подтвердить пользователя, четыре <include> зависимости, и отношения между действующими лицами и вариантами использования.



Рис. Л4.5 - Диаграмма вариантов использования для ATM

Варианты использования: Снять наличные, Перевести деньги со счета, Положить деньги на счет и посмотреть баланс - требуют включения идентификации клиента в системе. Это поведение может быть выделено в новый вариант использования включения, называемый Аутентификация пользователя. Базовые варианты использования не зависимы от метода, используемого для идентификации. Поэтому он инкапсулируется (скрывается) в варианте использования включения. С точки зрения базовых вариантов использования не имеет значения - производится ли идентификация с помощью магнитной карты или сканированием сетчатки глаза. Они только зависят от результата выполнения варианта использования Аутентификация пользователя. Вариант использования "Произвести оплату" предоставляет кредитной системе информацию об оплате по кредитной карточке. Вариант использования "Снять деньги" дополняется вариантом "Ускоренное снятие денег". Заметим, что на рисунке изображен только частный взгляд на модель вариантов использования. Полная модель вариантов использования также включает описания каждого действующего лица, каждого варианта использования, и спецификации на каждый вариант использования.

Л4.6.2 Диаграммы взаимодействия

**4.6.2.1** Диаграммы взаимодействия (interaction diagrams) описывают поведение взаимодействующих групп объектов. Как правило, диаграмма взаимодействия охватывает поведение объектов в рамках только одного варианта использования. На такой диаграмме отображается ряд объектов и те сообщения, которыми они обмениваются между собой [6].

В системе SEA существует 4 вида диаграмм взаимодействия [4]:

* Timing Diagrams (временные диаграммы)
* Sequence Diagrams (диаграммы последовательности)
* Interaction Overview Diagrams (диаграммы обзора взаимодействия или диаграммы кооперации)
* Communication Diagrams (диаграммы коммуникации)

На диаграмме взаимодействия (Interaction) отображают один из процессов обработки информации в варианте использования. В варианте использования "Снять деньги" из примера ATM имеется несколько альтернативных потоков. Это значит, что для данного варианта использования нужно создать несколько диаграмм Взаимодействия. В результате на одной диаграмме будет показано, что происходит, когда все в порядке. На других будет отображен ход событий в альтернативных потоках: что произойдет, если клиент введет неправильный идентификационный номер, если денег на его счету меньше, чем он хочет снять, и т.д.

Диаграммы взаимодействия визуализируют практически те же детали, что уже были описаны в потоке событий, однако представляют их в форме, более удобной для разработчика. Главное здесь – объекты, которые должны быть созданы для реализации функциональных возможностей, заложенных в вариант использования.

**4.6.2.2** Подобно диаграммам последовательности, диаграммы коммуникации (Communication Diagrams) отображают поток событий в конкретном сценарии варианта использования. Главная особенность диаграммы коммуникации заключается в возможности графически представить не только последовательность взаимодействия, но и все структурные отношения между объектами, участвующими в этом взаимодействии. Прежде всего, на диаграмме коммуникации в виде прямоугольников изображаются участвующие во взаимодействии объекты, содержащие имя объекта, его класс и, возможно, значения атрибутов. Далее, как и на диаграмме классов, указываются ассоциации между объектами в виде различных соединительных линий. При этом можно явно указать имена ассоциации и ролей, которые играют объекты в данной ассоциации. Дополнительно могут быть изображены динамические связи – потоки сообщений. Они представляются также в виде соединительных линий между объектами, над которыми располагается стрелка с указанием направления, имени сообщения и порядкового номера в общей последовательности инициализации сообщений. В отличие от диаграммы последовательности, на диаграмме коммуникации изображаются только отношения между объектами, играющими определенные роли во взаимодействии, а последовательность взаимодействий и параллельных потоков определяется с помощью порядковых номеров [2]. На рисунке Л4.6 приведена диаграмма коммуникации, описывающая, как клиент снимает со счёта 20$.

Для создания диаграммы последовательности необходимо кликнуть правой кнопкой по варианту использования "Снять деньги" на диаграмме прецедентов, далее Add (Добавить)-> Communication Diagram (диаграмма коммуникации). Для создания эктора "Клиент" необходимо перетянуть эктора (из диаграммы прецедентов) из проводника на диаграмму. Для добавления элементов используется UML Toolbox. Для добавления устройств (устройство чтения карточки, кассовый аппарат) на диаграммах последовательностей используется элемент Entity; для обозначения пользовательского интерфейса (экран АТМ) – Boundary, для счета – Lifeline. Для добавления сообщений между объектами необходимо сначала добавить связи ассоциации, а затем добавить сообщение через контекстное меню связи (Add Message). Для задания последовательности сообщений и управления ей, необходимо использовать Diagram->Sequence Communication Messages, для создания новой группы сообщений – поставить галочку Start New Group в настройках сообщения.

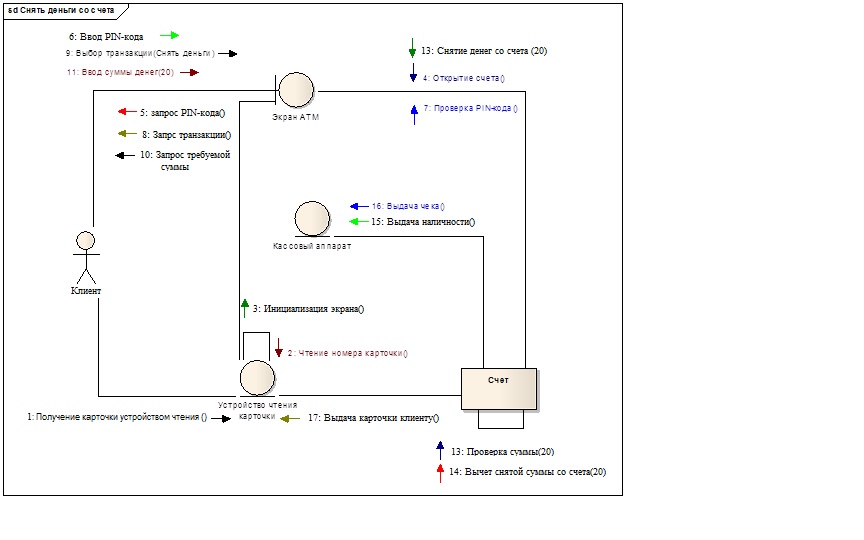


Рис. Л4.6 - Диаграмма коммуникациидля прецедента

снятия клиентом денег

**4.6.2.3** Подобно диаграммам коммуникации, диаграммы последовательности (Sequence Diagrams) отображают поток событий в конкретном сценарии варианта использования, но в них больше внимания уделяется именно времени. Пример сценария снятия 20$ со счета (при отсутствии таких проблем, как неправильный идентификационный номер или недостаток денег на счету) показан на рисунке Л4.7 [8].



Рис. Л4.7 - Диаграмма последовательности для прецедента снятия клиентом денег

Для создания диаграммы последовательности необходимо выполнить те же действия, что и для диаграммы коммуникации, только выбрать Sequence Diagram. Элементы добавлять стоит те же, что и на диаграмму коммуникации, т.е. перетягивать их из браузера проекта. Для добавления сообщения между элементами используется связь Message или Self-Message (для сообщения, отправляемого устройством самому себе), либо, выделив необходимую жизненную линию, можно потянуть за маленькую стрелочку, появившуюся рядом с выделением.

Таким образом, диаграмма последовательности иллюстрирует последовательность действий, реализующих вариант использования “Снять деньги со счета”. Глядя на эту диаграмму, пользователи знакомятся со спецификой своей работы. Аналитики видят последовательность (поток) действий, разработчики – объекты, которые надо создать, и их операции. Специалисты по контролю качества поймут детали процесса и смогут разработать тесты для их проверки. Таким образом, диаграммы последовательности полезны всем участникам проекта.

Из диаграммы коммуникациилегко понять поток событий и отношения между объектами, однако труднее уяснить последовательность событий. Именно поэтому для сценария дополнительно создают диаграммы последовательности. [2]

Л4.6.3 Диаграмма состояний.

**4.6.3.1** Главное предназначение диаграммы состояний (State Machine Diagram) – описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла. Чаще всего диаграммы состояний используются для описания поведения отдельных экземпляров классов (объектов), но они также могут быть применены для спецификации функциональности других компонентов моделей, таких как варианты использования, актеры, подсистемы, операции и методы.

Диаграмма состояний по существу 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.

Автомат (state machine) в языке UML представляет собой некоторый формализм для моделирования поведения элементов модели и системы в целом. Автомат описывает поведение отдельного объекта в форме последовательности состояний, которые охватывают все этапы его жизненного цикла, начиная от создания объекта и заканчивая его уничтожением. Каждая диаграмма состояний представляет некоторый автомат.

В языке UML под состоянием понимается абстрактный метакласс, используемый для моделирования отдельной ситуации, в течение которой имеет место выполнение некоторого условия. Состояние на диаграмме изображается прямоугольником со скругленными вершинами.

Начальное состояние представляет собой частный случай состояния, которое не содержит никаких внутренних действий. В этом состоянии находится объект по умолчанию в начальный момент времени. Графически начальное состояние в языке UML обозначается в виде закрашенного кружка, из которого может только выходить стрелка, соответствующая переходу.

Конечное (финальное) состояние представляет собой частный случай состояния, которое также не содержит никаких внутренних действий . В этом состоянии будет находиться объект по умолчанию после завершения работы автомата в конечный момент времени. Графически конечное состояние в языке UML обозначается в виде закрашенного кружка, помещенного в окружность, в которую может только входить стрелка, соответствующая переходу.

Простой переход (simple transition) 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. На диаграмме состояний переход изображается сплошной линией со стрелкой, которая направлена в целевое состояние.

**4.6.3.2** Рассмотрим пример диаграммы состояний для моделирования поведения банкомата (рис. Л4.8) [8]. Чтобы создать эту диаграмму, необходимо выполнить те же действия, что и для всех остальных диаграмм, только выбрать Statechart. Элементы добавлять стоит те же, что и на диаграмму коммуникации, т.е. перетягивать их из браузера проекта.

Сначала добавляем точки входы и выхода данных (Initial и final), и необходимое число состояний (state) используя UML Toolbox. Связываем элементы с помощью коннектора Transition. Эта связь показывает переход от одного состояния к другому. Для добавления сообщения к связи, необходимо использовать контекстное меню (Add Message), для добавления ограничения связи, например связь "Выбор Суммы", необходимо использовать поле Guard закладки Constraints настроек связи; а в случае связи "кредит превышен", используется еще и поле Effect.

Следует заметить, что в разрабатываемой модели диаграмма состояний является единственной и описывает поведение системы управления банкоматом в целом. Главное достоинство данной диаграммы состояний – возможность моделировать условный характер реализации всех вариантов использования в форме изменения отдельных состояний разрабатываемой системы. Иногда разработку диаграммы состояний, особенно в условиях дефицита времени, отпущенного на выполнение проекта, опускают, т.к. часто происходит дублирование информации, представленной на диаграммах кооперации и последовательности.



Рис. Л4.8 - Диаграмма состояний для прецедента снятия клиентом денег

Л4.6.4 Диаграмма деятельности

При моделировании поведения проектируемой или анализируемой системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и логической реализации выполняемых системой операций. Для моделирования процесса выполнения операций в языке UML используются так называемые диаграммы деятельности (Activity Diagram). Применяемая в них графическая нотация во многом похожа на нотацию диаграммы состояний, поскольку на диаграммах деятельности также присутствуют обозначения состояний и переходов.

Самым большим достоинством диаграмм деятельностей является поддержка параллелизма. Благодаря этому они являются мощным средством моделирования потоков работ и, по существу, параллельного программирования [2]. Самый большой их недостаток заключается в том, что связи между действиями и объектами просматриваются не слишком четко.

Рассмотрим пример диаграммы деятельности для прецедента «Снять деньги» (рисунок Л4.9) [8].

****

Рис. Л4.9 - Диаграмма деятельности для прецедента «Снять деньги»

Для создания диаграммы деятельности необходимо кликнуть правой кнопкой по варианту использования "Снять деньги" на диаграмме прецедентов, далее Add (Добавить)-> Activity Diagram (диаграмма деятельности).

Основные элементы:

* Action – представляет процесс либо какое-то преобразование в системе. Может являться дочерним элементом деятельности (Activity), но в отличие от этого элемента не может подвергаться декомпозиции
* Activity – также, как и действие представляет процесс либо какое-то преобразование в системе, может быть композитным элементом, который можно подвергать декомпозиции или разбиению на более мелкие действия. Для создания композитного элемента выбрать Advanced->Composite Element из контекстного меню элемента.
* Decision – это элемент, определяющий дальнейшее направление с учетом выполнения определенного условия. Если условие выполняется, данные идут по одному пути, если нет – то по другому.
* Fork\Join – используется для разделения\объединения нескольких потоков.
* Control flow – коннектор, используемый в диаграммах деятельности. Эти связи перемещают поток между узлами деятельности, показывая, на какой узел необходимо перейти по завершении работы текущего. Для указания значения условия при выходе из элемента Desicion, необходимо использовать поле Guard в закладке Constraints настроек связи.

Л4.6.5 Диаграмма классов

Диаграмма классов (Class Diagram) определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами.

Диаграмма классов для варианта использования «Снять деньги» показана на рисунке Л4.10 [8]. Для создания такой диаграммы необходимо кликнуть правой кнопкой по варианту использования "Снять деньги" на диаграмме прецедентов, далее Add (Добавить)-> Statechart (можно и любую другую), в появившемся списке выбрать Class Diagram.

В моделируемом процессе задействованы четыре класса: Card Reader (устройство чтения карточек), Account (счет), ATM Screen (экран АТМ) и Cash Dispenser (кассовый аппарат). Для добавления нового класса необходимо выбрать элемент Class на панели инструментов UML. Каждый класс на диаграмме выглядит в виде прямоугольника, разделенного на три части. В первой содержится имя класса, во второй – его атрибуты. В последней части содержатся операции класса, отражающие его поведение (действия, выполняемые классом). Чтобы добавить атрибуты и операции, нужно воспользоваться закладкой Details настроек класса.



Рис. Л4.10 - Диаграмма классов для прецедента «Снять деньги»

Связывающие классы линии отражают взаимодействие между классами. Так, класс Account связан с классом ATM Screen (экран АТМ), потому что они непосредственно сообщаются и взаимодействуют друг с другом. Класс Card Reader (устройство для чтения карточек) не связан с классом Cash Dispenser (кассовый аппарат), поскольку они не сообщаются друг с другом непосредственно.

## Л4.7 Варианты заданий по ЛР4

Для приобретения практических навыков проектирования автоматизированных систем (АС) и их ПО на основе языка UML в среде CASE-средства SEA студент должен заранее до начала лабораторной работы получить у преподавателя задание на ее выполнение. Задание содержит описание требующего автоматизации бизнес-процесса (процессов) из некоторой предметной области, например:

* процессов отдела кадров предприятия;
* процессов бухгалтерского подразделения предприятия;
* процессов продажи компьютеров и ПО компьютерной фирмы;
* процессов разработка ПО компьютерной фирмы;
* процессов агентства по торговле недвижимостью;
* процессов отдела по работе с претензиями клиентов фирмы;
* процессов кафедры учреждения образования;
* процессов поликлиники;
* процессов библиотеки;
* и т.п.

Варианты описаний бизнес-процессов приведены в приложении Б.   
В качестве варианта допускается использование бизнес-процесса, автоматизируемого в рамках курсовой работы по дисциплине «Проектирование автоматизированных систем» (ПАС).

## Л4.8 Порядок выполнения ЛР4

Рекомендуется следующий порядок выполнения работы ЛР4:

1. Изучить теоретический материал по языку UML (п.п. Л4.2, Л4.3).
2. Ознакомиться с программным средством SEA (п.п. Л4.4-Л4.6).
3. На основе анализа заданного бизнес-процесса сформулировать задачу его автоматизации.
4. Построить в среде SEA модели (проектные диаграммы UML), описывающие объект автоматизации и проектируемую систему на верхнем уровне детализации (согласно п.Л4.6).
5. Дополнить диаграммы текстовыми комментариями и пояснениями.
6. Сохранить полученные модели в репозитории SEA и на внешнем носителе информации.
7. Оформить отчет по выполненной работе в соответствии с правилами, описанными в п. Л4.9.
8. Ответить на контрольные вопросы.
9. Сдать отчет на проверку преподавателю и защитить работу.

## Л4.9 Правила оформления отчета по ЛР4

Отчет по лабораторной работе ЛР4 должен содержать следующие разделы:

1. Титульный лист.
2. Цель работы.
3. Краткое описание бизнес-процесса, полученного в задании по лабораторной работе.
4. Краткое описание постановки задачи автоматизации бизнес-процесса.
5. Построенные диаграммы UML и их описание.
6. Выводы по работе.

Образец оформления титульного листа представлен ниже в Приложении В.

## Л4.10 Контрольные вопросы по ЛР4

1. В чем состоит суть объектно-ориентированного подхода к проектированию автоматизированных систем и их ПО?
2. Дайте общую характеристику языка UML.
3. Что такое «Сущность», какие они бывают и как изображаются на диаграммах UML?
4. Что такое «Отношение», какие они бывают и как изображаются на диаграммах UML?
5. Что такое «Диаграмма UML», какие они бывают и каково их назначение в процессе проектирования автоматизированных систем?
6. Охарактеризуйте правила и общие механизмы языка UML.
7. Назовите основные характеристики и возможности CASE-средства проектирования Sparx Enterprise Architect.
8. Как организован интерфейс пользователя в среде Sparx Enterprise Architect?
9. Как создается новый проект в среде Sparx Enterprise Architect?
10. Какие типы шаблонов моделей существуют в Sparx Enterprise Architect?
11. Как строятся диаграммы прецедентов?
12. Какие бывают и как строятся диаграммы взаимодействия?
13. Как строятся диаграммы состояний?
14. Как строятся диаграммы классов?

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Арлоу, Д. UML 2 и унифицированный процесс. Практический объектно-ориентированный анализ и проектирование / Д. Арлоу, А. Нейштадт. – СПб.: Символ Плюс, 2007.
2. Бабич, А.В. UML: Первое знакомство / А.В. Бабич. – М.: БИНОМ. Лаборатория знаний, 2008.
3. Вендров, A.M. Проектирование программного обеспечения экономических информационных систем: учебник / A.M. Вендров. – М.: Финансы и статистика, 2004.
4. ISO/IEC 19501:2009 «OMG Unified Modeling Language™ (OMG UML), Infrastructure. Version 2.2» (англ.) (pdf). - Object Managemend Group (OMG), 2009.
5. Официальный сайт Sparx systems (англ.) [Электронный ресурс]. – Режим доступа: [www.sparxsystems.com](http://www.sparxsystems.com) свободный. – Загл. с экрана.
6. Леоненков, А.В. Объектно-ориентированный анализ и проектирование с использованием UML и IBM Rational Rose / А.В. Леоненков. – М.: Бином, 2006..
7. Ларман, К. Применение UML и шаблонов проектирования. 2-е издание. : Пер. с англ. – М. : Издательский дом "Вильяме", 2004.
8. Пашкевич, А.П. Современные технологии программирования : Метод. пособие / А.П. Пашкевич, О.А. Чумаков. – Минск: БГУИР, 2007.

# ПРИЛОЖЕНИЕ А

Содержание пунктов меню CASE-системы Sparx Enterprise Architect

Таблица А.1 - Содержание меню ‘Файл’ (File)

|  |  |
| --- | --- |
| Опция меню | Назначение и горячая клавиша |
| New Project | Создать новый проект (CTRL + N) |
| Open Project | Открыть проект (CTRL + O) |
| Open Source File | Открыть исходный файл (CTRL + ALT + O) |
| Close Project | Закрывает открытый проект |
| Save Project As | Сохраняет текущий проект с новым именем или создает ярлык на рабочем столе для этого проекта |
| Reload Current Project | Перезагружает текущий проект (используется в многопользовательском режиме, чтобы обновить окно проводника проекта) (CTRL + SHIFT + F11) |
| Page Setup | Сконфигурировать настройки страницы для вывода на печать |
| Print Setup | Настроить принтер |
| Print Preview | Предварительный просмотр страницы для вывода на печать |
| Print | Распечатать текущую диаграмму (CTRL + P) |
| Recent Files List | Список из максимум восьми часто загружаемых проектов |
| Exit | Выход из Enterprise Architect |

Таблица А.2 - Содержание меню ‘Редактировать’ (Edit)

|  |  |
| --- | --- |
| Опция меню | Назначение и горячая клавиша |
| Undo | Отмена последнего действия (некоторые действия не могут быть отменены) |
| Redo | Перевыполняет последнее отмененное действие |
| Copy | Копирует текущее выделение в буфер обмена |
| Add to Project Clipboard | Добавляет текущий элемент в буфер обмена SEA (CTRL + SPACE) |
| Clear Project Clipboard | Удаляет все элементы из буфера обмена |
| Paste Element(s) | Вставляет элементы из буфера обмена SEA в текущую диаграмму (см. таблицу А.3) |
| Select All | Выделяет все элементы на текущей диаграмме |
| Select By Type | Позволяет выбрать, какой тип элементов выделить |
| Clear Selection | Убирает выделение всех элементов |
| Find in Project | Позволяет выполнить поиск во всем проекте по определенным фразам и словам (CTRL+F) |
| Bookmark Selected | Помечает выбранные элементы. Если выбранный элемент уже помечен, убирает пометку (SHIFT + SPACE) |
| Clear All Bookmarks | Убирает все пометки со всех элементов |
| Delete Selected Element(s) | Удаляет выбранные элементы с диаграммы (CTRL + D) |

Таблица А.3 - Содержание подменю ‘Paste Element(s)’ меню ‘Редактировать’ (Edit)

|  |  |
| --- | --- |
| Опция меню | Назначение и горячая клавиша |
| As Link | Вставляет элемент из буфера, в виде ссылки на этот элемент (SHIFT+INSERT) |
| As New | Вставляет элемент из буфера, в виде нового элемента (CTRL + SHIFT + V) |
| Paste Image from Clipboard | Вставляет элемент из буфера в диаграмму в виде метафайла (рисунка) столько раз, сколько это необходимо. |

Таблица А.4 - Содержание меню ‘Просмотр’ (View)

|  |  |
| --- | --- |
| Опция меню | Назначение и горячая клавиша |
| Project Browser | Показывает\убирает проводник проекта (Project Browser) (ALT + 0) |
| Properties | Показывает\убирает окно свойств (Property) (ALT + 1) |
| System | Показывает\убирает окно системы (System window) (ALT + 2) |
| Testing | Показывает\убирает окно тестирования (Testing Window) (ALT + 3) |
| Maintenance | Показывает\убирает окно обслуживания (Maintenance) (ALT + 4) |
| Toolbox | Показывает\убирает Панель инструментов UML (Enterprise Architect UML Toolbox) (ALT + 5) |
| Resources | Показывает\убирает окно ресурсов (Resources) (ALT + 6) |
| Source Code | Показывает\убирает окно просмотрщика исходного текста (Source Code Viewer) (ALT + 7) |
| Debug Workbench | Показывает\убирает окно отладки (Debug Workbench) (ALT + 8) |
| Tasks Pane | Показывает\убирает окно быстрой справки (Task Pane) (CTRL + SHIFT + 9) |
| Notes | Показывает\убирает окно записок (Notes) (CTRL + SHIFT + 1) |
| Hierarchy | Показывает\убирает окно иерархии (Hierarchy Window) (CTRL + SHIFT + 4) |
| Tagged Values | Показывает\убирает окно помеченных значений элементов (Tagged Values) (CTRL + SHIFT + 6) |
| Project Management | Показывает\убирает окно управления проектом (Project Managment) (CTRL + SHIFT + 7) |
| Output | Показывает\убирает окно выходных данных (Output) (CTRL + SHIFT + 8) |
| More Windows | Еще больше окон (см. таблицу А.5) |
| Toolbars | Добавляет\убирает дополнительные панели инструментов |
| Element List | Показывает текущую диаграмму в виде редактируемого списка (CTRL + ALT + R) |
| Model Search | Показывает окно генерации отчетов Model Search (CTRL + ALT + A) |
| Relationship Matrix | Показывает окно матрицы отношений (Relationship Matrix) между элементами диаграммы. |
| Discussion Forum | Показывает окно форума (Discussion Forum), предназначенного для обсуждения и разработки и прогресса проекта (CTRL + ALT + U) |
| Audit View | Вызывает окно аудита (Audit View), которое показывает информацию, которая была сохранена в процессе аудита |
| Show Grid | Показывает\убирает сетку |
| Snap to Grid | Располагает элементы в соответствии с сеткой |
| Visual Style | Позволяет выбрать и настроить внешний вид системы: от обычного Windows-стиля до современного Vista-стиля. |
| Visual Layouts | Позволяет загружать и сохранять пользовательские настройки внешнего вида системы (т.е. расположения окон, панелей инструментов и т.д.) |

Таблица А.5 - Содержание подменю ‘More Windows’ меню ‘Просмотр’ (View)

|  |  |
| --- | --- |
| Опция меню | Назначение и горячая клавиша |
| Element Browser | Вызывает окно проводника элементов (Element Browser), которое показывает все аспекты выбранных элементов. |
| Relationships | Вызывает окно отношений (Relationships), которое показывает все связи между выделенным элементом и остальными элементами. Оно позволяет производить быстрый обзор связей элемента в модели (CTRL + SHIFT + 2) |
| Rules and Scenarios | Окно правил и сценариев (Rules and Scenarios) показывает все требования, ограничения, сценария и т.п. ,связанные с элементом. Это окно предоставляет удобный способ быстро просматривать, изменять и добавлять правила выбранному элементу (CTRL + SHIFT + 3) |
| Web Browser | Открывает встроенный веб-браузер (CTRL + ALT + W) |
| Pan and Zoom | Окно Pan and Zoom предоставляет вид диаграммы с высоты птичьего полета. Оно позволяет быстро перемещаться по большим диаграммам (CTRL + SHIFT + N) |

Таблица А.6 - Содержание меню ‘Проект’ (Project)

|  |  |
| --- | --- |
| Опция меню | Назначение и горячая клавиша |
| Add Package | Создает новый пакет (Package) (CTRL + W) |
| Add Diagram | Создает новую диаграмму в текущем пакете (CTRL + Y) |
| Add Element | Создает новый элемент в текущей диаграмме (CTRL + M) |
| Documentation | Генерирует различные виды документации (см. Таблицу А.7) |
| Source Code Engineering | Выполняет прямую и обратную разработку (forward and reverse engineering) кода с использованием языка на выбор (см. таблицу А.8) |
| Build and Run | Связывает проект с компилятором для построения, выполнения и отладки (см. таблицу А.9) |
| Database Engineering | Позволяет импортировать схему БД из источника данных ODBC, а также генерировать DDL-скрипты. |
| Model Transformations | Выполняет трансформацию элемента (CTRL + ALT + F) или пакета (CTRL + SHIFT + H) |
| Model Validation | Проверка правильности использования UML в элементе, модели, или всем пакете |
| Web Services | Позволяет использовать прямую и обратную разработку с использованием языка WSDL (Web Service Definition Language) |
| XML Schema | Позволяет использовать прямую и обратную разработку с использованием языка W3C XML Schema (XSD) |
| Import\Export | Позволяет управлять оаперациями экспорта/импорта данных |
| View Project Statistics | Просмотреть статистику проекта |

Таблица А.7 - Содержание подменю ‘Documentation’ меню ‘Проект’ (Project)

|  |  |
| --- | --- |
| Опция меню | Назначение и горячая клавиша |
| Rich Text Format Report | Генерирует отчет для выбранного пакета в формате RTF (А8) |
| HTML Report | Генерирует отчет для выбранного пакета в формате HTML (SHIFT + F8) |
| Diagrams Only Report | Генерирует отчет для выбранного пакета в формате RTF, содержащий только диаграммы (CTRL + SHIFT + F8) |
| Testing Report | Генерирует отчет о существующей тестовой документации модели в формате RTF |
| Issues | Генерирует отчет о проблемах модели в формате RTF |
| Glossary | Генерирует отчет глоссария модели в формате RTF |
| Implementation Details | Генерирует отчет о выполнении для текущего пакета |
| Dependency Details | Генерирует отчет о зависимостях для текущего пакета |
| Testing Details | Генерирует отчет о тестировании для текущего пакета |
| Resource and Tasking Details | Генерирует отчет о ресурсах для текущего пакета |

Таблица А.8 - Содержание подменю ‘Source Code Engineering’ меню ‘Проект’ (Project)

|  |  |
| --- | --- |
| Опция меню | Назначение и горячая клавиша |
| Generate Package Source Code | Сгенерировать исходный код для выбранного пакета (CTRL+ALT+K) |
| Synchronize Package Contents | Синхронизировать выбранный пакет с исходным кодом (CTRL+ALT+M) |
| Import Source Directory | Выполнить реверс инжиниринг каталога |
| Import Action Script Files | Импортировать код, написанный на Action Script с расширением .AS |
| Import C Files | Импортировать код, написанный на C с расширением .C или .H |
| Import C# Files | Импортировать код, написанный на C# с расширением .CS |
| Import C++ Files | Импортировать код, написанный на C++ с расширением .H, .HPP, или .HH |
| Import Delphi Files | Импортировать код, написанный на Delphi с расширением .PAS |
| Import Java Files | Импортировать код, написанный на Java с расширением .JAVA |
| Import PHP files | Импортировать код, написанный на PHP с расширением .PHP, .PHP4, INC |
| Import Python Files | Импортировать код, написанный на Python с расширением .PY |
| Import Visual Basic | Импортировать код, написанный на Visual Basic с расширением .FRM, .CLS, .BAS, .CTL |
| Import VB.Net Files | Импортировать код, написанный на VB.Net с расширением .VB |

Таблица А.9 - Содержание подменю ‘Build and Run’ меню ‘Проект’ (Project)

|  |  |
| --- | --- |
| Опция меню | Назначение и горячая клавиша |
| Package Build Scripts | Создать и настроить скрипты компилятора (SHIFT + 12) |
| Build | Построить приложение для скрипта (CTRL + SHIFT + F12) |
| Test | Выполнить тестирующий скрипт (CTRL + SHIFT +F12) |
| Run | Выполнить (CTRL + F12) |
| Deploy | Выполнить скрипт развертывания (CTRL + SHIFT + ALT + F12) |
| Debug Run | Выполнить скрипт запуска (с отладкой) (CTRL + ALT + F5) |

Таблица А.10 - Содержание меню ‘Диаграмма’ (Diagram)

|  |  |
| --- | --- |
| Опция меню | Назначение и горячая клавиша |
| Properties | Открывает диалог настроек для текущей диаграммы с заголовком <тип диаграммы> Diagram: <название диаграммы> (F5) |
| Layout Diagram | Настроить автоматическое построение диаграмм. Недоступно для диаграмм поведения (Behavioral diagrams) |
| Lock Diagram | Запрещает внесение изменений на диаграмме |
| Save | Сохраняет текущее положение элементов на диаграмме (CTRL + S) |
| Save as Image | Сохраняет диаграмму в виде растрового изображения – BMP, GIF или WMF (CTRL + T) |
| Save as UML Pattern | Сохранить текущую диаграмму в качестве UML шаблона |
| Copy Image to Clipboard | Скопировать текущую диаграмму в буфер обмена (CTRL + B) |
| Configure Swimlanes and Matrix | Добавить, изменить или удалить разделительный линии (swimlanes) или разделительную матрицу (swimlanes matrix) для текущей диаграммы |
| Set Visible Relations | Настроить видимость связей (CTRL + SHIFT + I) |
| Insert Property Note | Отображает настройки текущей диаграммы на самой диаграмме |
| Locate in Project Browser | Выделить текущую диаграмму в проводнике проекта (CTRL + SHIFT + G) |
| Set as User Default | Если включена защита диаграммы от редактирования, назначает текущую диаграмму в качестве диаграммы по умолчанию для данного пользователя. В следующий раз, когда данный пользователь откроет эту модель, первым, что он увидит, будет диаграмма по умолчанию. |
| Set as Model Default | Назначает текущую диаграмму в качестве диаграммы по-умолчанию для данной модели. В следующий раз, когда эта модель будет открыта, первым, что увидит пользователь, будет диаграмма по-умолчанию |
| Change Diagram Type | Сменить тип текущей диаграммы |
| Repeat Last Element | Создать такой же элемент, какой был создан до этого (SHIFT + T) |
| Repeat Last Connector | Создать такую же связь, какая была создана до этого (F3) |
| Zoom | Изменить масштаб текущей диаграммы |

Таблица А.11 - Содержание меню ‘Элемент’ (Element)

|  |  |
| --- | --- |
| Опция меню | Назначение и горячая клавиша |
| Properties | Открывает окно настроек текщего элемента |
| Add Tagged Value | Добавить выбранному элементу помеченное значение (Tagged Value) (CTRL + SHIFT + T) |
| Linked Document | Привязать элемент к документу RTF (CTRL + ALT + D) |
| Attributes | Просмотреть и редактировать атрибуты выбранного элемента (F9) |
| Operations | Просмотреть и редактировать операции выбранного элемента (F10) |
| Inline Features | Это субменю предоставляет различные возможности для редактирования элементов на диаграмме классов (Class diagram) (см. таблицу А.12) |
| Set Feature Visibility | Различные настройки, относящиеся к видимости выбранного элемента (CTRL + SHIFT + Y) |
| Advanced | Это субменю предоставляет различные возможности для настройки элементов модели |
| Rich Text Format (RTF) Report | Сгенерировать отчет для выбранного пакета в формате RTF |
| Source Code Engineering | Позволяет применять форвард и реверс программирование с использованием любого языка (см. таблицу А.13) |
| Open Source in External Editor | Открыть исходный код выбранного класса во внешнем редакторе для этого языка, установленному по-умолчанию. (CTRL + E или F 12) |
| Locate in Project Browser | Выделить выбранный элемент в проводнике проекта (Project Browser) (ALT + G - если элемент не выбран, выделяет диаграмму) |
| Find in Diagrams | Показывает все элементы данного типа на диаграмме |
| Appearance | Это субменю предоставляет различные возможности для настройки внешнего вида элементов модели, включая размеры, границы, цвета и даже изображения элементов |
| Alignment | Это субменю предоставляет возможности по выравниванию элементов друг относительно друга |
| Make Same | Это субменю предоставляет возможности по управлению шириной, высотой элементов и т.п. С его помощью можно выровнять размеры элементов друг относительно друга |
| Z Order | Это меню используется для расположения элементов по оси Z. Можно поместить выбранный элемент перед и за остальными элементами |
| Size | Это меню позволяет управлять размерами выделенных элементов, делать их шире, меньше, тоньше и т.п. по одному проценту |
| Move | Это меню позволяет двигать элементы вверх, вниз, влево, вправо по одному проценту |
| Space Evenly | Это меню позволяет сделать равномерные интервалы между элементами по горизонтали (ALT + -) или по вертикали (ALT + =) |

Таблица А.12 - Содержание подменю ‘Inline Features’ меню ‘Элемент’ (Element)

|  |  |
| --- | --- |
| Опция меню | Назначение и горячая клавиша |
| Edit Selected | Добавить элементу примечание или ограничение (F2) |
| View Properties | Открывает диалог, содержащий настойки выбранной детали элемента, или самого элемента |
| Insert New After Selected | Вставляет новую деталь в элемент, такого же типа, как и выделенная (CTRL + SHIFT + INSERT) |
| Add Attribute | Добавляет новый атрибут в элемент (CTRL + SHIFT + F9) |
| Add Operation | Добавляет новую операцию в элемент (CTRL + SHIFT + F10) |
| Add Other | Позволяет добавить в элемент специальную деталь, такую как, например, Maintenance features или Testing features (CTRL + F11) |
| Delete Selected from Model | Удаляет выбранную деталь из элемента (CTRL + SHIFT + DELETE) |

Таблица А.13 - Содержание подменю ‘Source Code Engineering’ меню ‘Элемент’ (Element)

|  |  |
| --- | --- |
| Опция меню | Назначение и горячая клавиша |
| Generate Current Element | Сгенерировать исходный код для выбранного элемента (CTRL + G или F11) |
| Synchronize Current Element | Синхронизировать выделенный класс с исходным кодом (CTRL + R или F7) |
| Batch Generate Selected Elemets(s) | Сгенерировать исходный код для группы выбранных элементов (SHIFT + F11) |
| Batch Synchronize Selected Elemets(s) | Синхронизировать выделенные элементы с исходным кодом (CTRL + R) |
| Open Source Directory | Открыть папку, содержащую исходный источник для выбранного элемента (CTRL + ALT + Y) |

Таблица А.14 - Содержание меню ‘Инструменты’ (Tools)

|  |  |
| --- | --- |
| Опция меню | Назначение и горячая клавиша |
| Spell Check Project | Проверяет орфографию во всем проекте (CTRL + F7) |
| Spell Check Current Package | Проверяет орфографию в выбранном пакете (CTRL + F7) |
| Spelling Language | Выбрать язык для проверки орфографии |
| Data Management | Позволяет управлять данными проекта |
| Manage .EAP File | Позволяет управлять файлами .EAP – сжимать, копировать, восстанавливать битые и т.п. |
| Export Reference Data | Экспортирует текущую модель в XML-файл |
| Import Reference Data | Импортирует XML-файл в текущую модель, обновляя и дополняя данные |
| Import Technology | Импортировать MDG технологию, которая сразу содержит необходимые настройки UML, шаблоны, картинки, значения и др |
| Generate MDG Technology file | Создать MDG технологию, которая сразу содержит необходимые настройки UML, шаблоны, картинки, значения и др., используя мастер |
| Wordpad | Открыть Wordpad |
| Windows Explorer | Открыть Windows Explorer |
| Customize | Настроить панели инструментов и горячие клавиши по своему вкусу |
| Options | Позволяет произвести настройку системы в целом. |

Таблица А.15 - Содержание меню ‘Настройки’ (Settings)

|  |  |
| --- | --- |
| Опция меню | Назначение и горячая клавиша |
| People | Показывает диалог People (люди), который позволяет настроить авторов, клиентов, ресурсы и т.п. для проекта |
| General Types | Позволяет настроить требования, ограничения, сценарии и т.п. для вашего проекта |
| Maintenance | Позволяет настраивать проблемные типы и типы для тестирования |
| Project Indicators | Позволяет определить индикаторы проекта которые используются в управлении ресурсами (Resource Managment) |
| Estimation Factors | Позволяет настроить, сколько времени и усилий требуется для разработки и развертывания проекта |
| UML | Позволяет настраивать UML |
| MDG Technologies | Позволяет загружать и использовать файлы MDG Technologies |
| Namespaces | Позволяет находить и удалять пробелы в названиях |
| Template Package | Позволяет настраивать и изменять директорию для шаблонов по-умолчанию |
| Local Paths | Позволяет настраивать все необходимые пути для системы |
| Auto Name Counters | Позволяет настроить автоматическое присвоение имен элементам |
| Code Datatypes | Позволяет добавлять, изменять и удалять типы данных для каждого из языков |
| Database Datatypes | Позволяет добавлять, изменять и удалять типы данных для каждого вида баз данных |
| Preprocessor Macros | Позволяет добавлять и удалять макросы |
| Code Generation Templates | Позволяет изменять шаблоны кодогенерации (которые используются при форвард инжиниринге), используя специальный редактор (Code Templates editor) (CTRL + SHIFT + P) |
| Images | Открывает менеджер рисунков, который позволяет добавлять на диаграммы внешние рисунки (кроме элементов UML) |
| Colors | Позволяет настроить цвета проекта |

# ПРИЛОЖЕНИЕ Б

Варианты индивидуальных заданий по лабораторной работе

**Бизнес-процесс 1 "Планирование закупок и размещение заказов поставщикам"**

*Общее описание бизнес-процесса*

Предприятие планирует закупки медикаментов. Планирование закупок осуществляется в Департаменте маркетинга, в группе маркетинга и планирования. Планирование закупок осуществляется следующим образом:

1. Менеджер группы планирования и маркетинга ежесуточно получает от контрагентов данные внешней и внутренней статистики продаж медикаментов в виде отчетов продаж.
2. Для планирования закупок медикаментов менеджер группы планирования и маркетинга еженедельно на основании статистики продаж осуществляется расчет потребности в товаре. В результате расчета формируется **Таблица потребностей в товаре**.
3. Определив количество и номенклатуру заказываемых товаров, менеджер отдела закупок приступает к анализу предложений поставщиков. Данный процесс осуществляется ежемесячно или по мере необходимости. Выбираются наиболее выгодные условия поставки. Для этого сравниваются цены поставщиков. Данные сведения берутся из прайс-листа для закупок. При выборе поставщика важно учесть предоставляемую отсрочку платежа. Данные сведения берутся из контрактов, отмеченных как приоритетные (действующие). В результате, формируется список поставщиков, каждой позиции присваивается признак основного и запасных поставщиков в порядке убывания приоритета.
4. Менеджер отдела закупок ежемесячно на основании **Таблицы потребностей в товаре** и списка выбранных поставщиков формирует графики поставок с указанием сроков и периодичности, но без количества поставки.
5. Ежемесячно после определения потребности в товаре менеджер группы логистики рассчитывает необходимое количество закупок. Необходимое количество закупок рассчитывается на основании фактических запасов на складе, необходимого минимального и максимального уровня запасов. Нормы минимального и максимального количества запасов устанавливаются в днях. При расчете необходимого количества закупки учитывается также время товара в пути. Таким образом, данный расчет должен обеспечить возможность бесперебойного наличия товара на складе. По результату расчетов формируется план заявок на месяц.
6. Затем в группе логистики ежедневно по плану заявок, графику поставок, прайс-листам поставщиков формируются заказы поставщикам.
7. Если предстоит сделать заказ импортному поставщику, то менеджер группы логистики рассчитывает затраты на сертификацию, создается отчет о затратах на сертификацию. Затраты на сертификацию проверяются на соответствие внутрифирменным нормам. Данная операция производится по мере необходимости.
8. Если затраты на сертификацию превышают внутрифирменные нормы, то менеджер группы логистики повторяет процесс формирования заказов поставщикам. Формируются новые заказы.
9. Ежедневно подготовленный заказ поставщику акцептуется, заказ должен подписать менеджер по логистике и директор Департамента маркетинга и управления товарными запасами.
10. Ежедневно менеджер группы логистики направляет заказ в отдел закупок. Менеджер отдела закупок направляет заказ поставщику.

**Бизнес-процесс 2 "Приходование товара"**

*Общее описание бизнес-процесса*

ЗАО "МЕД" располагает 10 складами, из которых один Центральный в Минске, другие в филиалах. Количество хранимой номенклатуры медикаментов от 1000 до 2000.

Склад фактически работает не с номенклатурой, а с сериями. Одной позиции номенклатуры может соответствовать несколько серий медикаментов.

Склад разбит на несколько зон хранения. Зоны хранения соответствуют правилам хранения тех или иных медикаментов.

Используются вложенные единицы измерения - упаковка (минимальная единица), блок (несколько упаковок), заводская коробка (несколько блоков).

На складе также хранится зарезервированный, недоступный для продажи товар.

Учет ТМЦ ведется в двух валютах - в рублях, валюте прихода.

Процесс приходования медикаментов на склад выглядит следующим образом:

1. Менеджер приемного отдела принимает товар по товарной накладной поставщика, проверяя номенклатуру, количество, посерийное соответствие, срок годности.
2. При полном соответствии фактически поступившего товара, товару указанному в товарно-транспортной накладной и заказе поставщику, менеджер приемного отдела передает документы менеджеру отдела закупок. В противном случае осуществляется процесс выявления виновных лиц и предъявление претензий.
3. Менеджер отдела закупок проверяет соответствие поставки заказу по номенклатуре, количествам и ценам и на основании товарной накладной поставщика формирует приходную накладную, отражая в базе данных количество и учетную цену поступившего товара. При формировании приходной накладной создается проводка Д41-К60. Далее в работу включаются менеджеры отделов сертификации и маркетинга.
4. Менеджер отдела сертификации по товарно- транспортной накладной проверяет наличие серий в справочнике. При необходимости справочник серий пополняется.
5. Менеджером отдела сертификации осуществляется процесс сертификации. Процесс сертификации в данном случае рассматривается и как процесс приходования сертификатов-документов на медикаменты, и как процедуры сертификации с целью получения документов-сертификатов.
6. Менеджер учетного отдела при приходовании ТМЦ по товарно-транспортной накладной разбивает каждую номенклатурную позицию по сериям с указанием срока годности.
7. Параллельно с работой менеджера по сертификации, после отражения в базе данных количества товара менеджером отдела закупок, менеджер отдела маркетинга, используя товарно-транспортную накладную, определяет базовую цену продажи и указывает ее в карточке товара.
8. Размещение товара по местам хранения осуществляется менеджером склада в соответствии с Планом расстановки продукции по местам хранения. Место хранение заносится в карточку товара.

**Бизнес-процесс 3 "Продажи"**

Бизнес-процесс выглядит следующим образом:

1. Менеджер отдела продаж ежедневно получает от клиента **Заказ** на конкретную номенклатурную единицу медикаментов. В **Заказе** номенклатурных единиц клиент указывает желаемую отсрочку платежа.
2. При получении **Заказа** менеджер отдела продаж по справочнику лицензий проверяет наличие у клиента действующей лицензии на право реализации медикаментов. При отсутствии лицензии продажа медикаментов клиенту не производится. Наличие лицензии проверяется по мере необходимости.
3. Менеджер отдела продаж ежедневно проверяет наличие необходимого количества заказанных медикаментов на складе.
4. Если медикаментов недостаточно для выполнения заказа, то менеджер отдела продаж размещает **Заказ** в реестре "**неудовлетворенный спрос**". Затем менеджер ежедневно проверяет возможность выполнения **Заказа**, размещенного в реестре "**неудовлетворенный спрос**".
5. При наличии у клиента необходимой лицензии и достаточном количестве товара на складе в отделе продаж на основании **Заказа** и договора формируется **Заявка на номенклатурные единицы**. Заявки формируются ежедневно.
6. Ежедневно на основании **Заявки** менеджер отдела продаж осуществляет резервирование товара.
7. Менеджер отдела продаж ежедневно контролирует кредитный лимит, дебиторскую задолженность потенциальных покупателей.
8. Если кредитный лимит и дебиторская задолженность не превышают допустимых значений, то **Заявка** передается на склад в **Учетно-операционный отдел**.
9. При превышении кредитного лимита или наличия просроченной дебиторской задолженности свыше допустимого количества дней менеджер отдела продаж заявку в Учетно-операционный отдел не передает, процесс продаж приостанавливается, осуществляются переговоры с клиентом.
10. Менеджер учетно-операционного отдела, получив **Заявку**, ежедневно осуществляет подборку номенклатурных единиц.
11. Менеджер учетно-операционного отдела ежедневно формирует упаковочные листы для вложения их в каждый ящик.
12. Менеджером учетно-операционного отдела ежедневно формируются для клиента следующие документы: счет, расходная накладная, счет-фактура.
13. При фактической отгрузке товара со склада осуществляется его списание. Списание медикаментов осуществляется по расходной накладной и сопровождается формированием проводки **Д62-К41**.

**Бизнес-процесс 4 "Взаиморасчеты с клиентами"**

Бизнес-процесс выглядит следующим образом:

1. Менеджер отдела продаж до 10 раз в день отгружает товары клиентам в соответствии с договорами и Приказом по кредитной линии. Одновременно с отгрузкой товара менеджер отдела продаж выставляет счет клиенту. Счет регистрируется в реестре счетов.
2. По факту произведенной отгрузки менеджер отдела продаж делает запись в журнале отгрузок и оплат, тем самым, фиксируя задолженность клиента.
3. Бухгалтер компании ежедневно получает и обрабатывает выписки с расчетных счетов банков. Бухгалтер на основании банковской выписки определяет оплаченные счета и делает отметку об оплате счета в реестре счетов.
4. Менеджер отдела продаж ежедневно контролирует поступление платежей от клиентов, проверяя допустимый срок оплаты счета.
5. Если платежи по счету на расчетный счет компании не поступили, и срок оплаты счета истек, то менеджер отдела продаж блокирует отгрузку товара клиенту. Если клиент оплатил счет, то менеджер вносит сведения об оплате в **Журнал отгрузок и оплат**.
6. Бухгалтер в конце каждого месяца выводит сальдо взаиморасчетов с клиентами

**Бизнес-процесс 5 "Взаиморасчеты с поставщиками"**

Бизнес-процесс выглядит следующим образом:

1. Менеджер отдела закупок ежедневно получает от поставщика медикаментов счет на оплату, регистрирует его в реестре счетов поставщиков и передает счет поставщика бухгалтеру.
2. Бухгалтер на основании счета поставщика ежедневно формирует платежное поручение на оплату и передает платежное поручение в банк.
3. Бухгалтер на основании выписки с расчетного счета банка делает отметку об оплате счета в реестре счетов поставщика.
4. Менеджер отдела закупок при поступлении товара и (или) при оплате делает запись в **Журнале поступлений и оплат**.
5. Бухгалтер в конце каждого месяца выводит сальдо взаиморасчетов с клиентами.

# ПРИЛОЖЕНИЕ В

Образец оформления титульного листа

**Министерство образования Республики Беларусь**

**Учреждение высшего образования**

**«Белорусский государственный университет**

**информатики и радиоэлектроники»**

**Факультет информационных технологий и управления**

**Кафедра информационных технологий автоматизированных систем**

**Лабораторная работа № 4 по дисциплине**

**ПРОЕКТИРОВАНИЕ АВТОМАТИЗИРОВАННЫХ СИСТЕМ**

**на тему**

**«Объектно-ориентированное моделирование**

**средствами языка UML»**

**Вариант задания \_\_\_**

**Выполнил студент группы \_\_06\_\_**

**Фамилия И.О. студента**

**Проверил преп. каф. ИТАС**

**Фамилия И.О. преподавателя**

**Минск 20\_\_\_**