### ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Информационные технологии развиваются стремительно. В условиях жесткой конкуренции на рынке и растущих запросов пользователей создание информационных систем стало очень сложной задачей. Системы стали настолько велики, что физических и умственных возможностей и способностей человека уже просто не достаточно для того, чтобы спроектировать сложную систему за один шаг, и даже для того, чтобы просто представить, вообразить все возможности и потребности проектируемой системы, ее архитектуру и программное обеспечение. Однако замечено, что визуальная информация воспринимается наиболее успешно и полно. На помощь проектировщику сложных систем пришло визуальное моделирование и проектирование.

Инструментальные средства разработки программного обеспечения – это междисциплинарный курс, целью которого является овладение теоретическими и практическими знаниями по моделированию информационных систем, знаниями принципов и методов их построения, получение теоретических знаний и практических навыков работы с современными CASE-средствами, методологиями проектирования информационных систем. Овладение навыками самостоятельного использования соответствующих инструментальных программных систем позволяет осуществить проектирование информационных систем. Главными преимуществами CASE-технологии по сравнению с другими способами моделирования являются:

- создание модели системы в приемлемые сроки;

- сокращение затрат связанных с процессом проектирования;

- возможность анализа полученных данных до начала процесса непосредственного создания системы (тем самым появляется возможность «предотвращения» глобальных ошибок);

-возможность применения «готовых» разработок в соответствии со своими требованиями (стандартные программные продукты и инструменты, относящиеся к группе CASE-средств).

Предлагаемое учебно-методическое пособие «CASE-технологии проектирования информационных систем в среде StarUML» посвящено рассмотрению основных приемов визуального моделирования систем с помощью UML и предназначено для студентов для аудиторных и самостоятельных занятий. В пособии описываются основные элементы нотации диаграмм UML, на конкретных примерах рассматривается процесс проектирования информационных систем с применением программной платформы StarUML, приводятся некоторые приемы и способы создания моделей системы: поиск классов, их атрибутов и операций, поиск объектов системы и др. Этапы создания визуальной модели сопровождаются иллюстрированными инструкциями. Пособие составлено с учетом требований ФГОС третьего поколения.

Актуальность составления данного пособия обусловлена потребностью активного использования CASE-средств как основы автоматизации проектирования. Пособие направлено на повышение мотивации обучающихся, развитие гибкого логического и пространственного мышления обучающихся, развитие профессиональных компетенций обучающейся молодежи.

В данном пособии предлагаются 14 лабораторных работ по разделу «CASE-технологии». Лабораторные работы, приведенные в учебно-методическом пособии, включают в себя следующие элементы: название темы, цель занятия, теоретическую часть, практическую часть (задания, порядок выполнения работы) и контрольные вопросы. В теоретической части приводится структурированный материал, необходимый для подготовки студентов к лабораторным работам. Практическая часть представляет собой логично выстроенный ход работы, включающий задания, этапы выполнения работы и примеры реализации заданий. Вопросы к каждой лабораторной работе позволяют провести контроль и самоконтроль обучающихся.

В учебно-методическом пособии содержатся задания для освоения основных этапов визуального моделирования информационной системы. Моделирование с помощью UML осуществляется поэтапным построением ряда диаграмм, каждая из которых отражает какую-то часть или сторону системы либо ее замысла.

Основные диаграммы UML, достаточные для полного моделирования системы поэтапно рассматриваются в учебном пособии:

вариантов использования (usecasediagram);

классов (class diagram);

кооперации (collaboration diagram);

последовательности (sequence diagram);

состояний (statechart diagram);

деятельности (activity diagram);

компонентов (component diagram);

развертывания (deployment diagram).

В данном пособии рассматриваются основные элементы нотации диаграмм и принципы их построения. Учебно-методическое пособие содержит лабораторные работы, которые обеспечивают формирование базовых умений и навыков проектирования информационных систем. В результате выполнения лабораторных работ, обучающиеся получат практический опыт визуального моделирования информационных систем в среде StarUML. Выбор содержания и объем конкретной лабораторной работы обусловлен сложностью учебного материала для усвоения, междисциплинарными связями и учетом значения конкретной лабораторной работы для приобретения обучающимися соответствующих умений и компетенций, предусмотренных ФГОС. Методика выполнения каждой лабораторной работы определяется моделью соответствующей задачи, решаемой студентом на занятии по заданию преподавателя. В лабораторных работах, приведенных в пособии, содержатся как задания с подробными указаниями к выполнению, так и задания без алгоритма работы. Таким образом, предлагаемые практические работы можно использовать для индивидуальной работы студентов на занятиях под руководством преподавателя, а также для самостоятельной работы обучающихся.

**Учебно-методическое пособие**

**«CASE-технологии проектирования**

**информационных систем**

**в среде StarUML»**

СОДЕРЖАНИЕ

Пояснительная записка 3

[Лабораторная работа №1](#_Toc497220372)[[Создание диаграмм прецедентов (вариантов) использования](#_Toc497220372)](#_Toc497220373) [5](#_Toc497220372)

[Лабораторная работа №2 Создание диаграмм последовательностей 16](#_Toc497220374)

[Лабораторная работа №3 Создание кооперативной диаграммы 21](#_Toc497220376)

[Лабораторная работа №4 Создание диаграммы состояний 27](#_Toc497220378)

[Лабораторная работа №5 Создание диаграмм активности (деятельности) 35](#_Toc497220381)

[Лабораторная работа №№6-7 Создание диаграмм классов 42](#_Toc497220383)

[Лабораторная работа № 8-9 Работа с инструментальными средствами, поддерживающими методологию объектно-ориентированного моделирования 55](#_Toc497220385)

[Лабораторная работа №№ 10-11 Работа с CASE–средствами проектирования программного обеспечения 71](#_Toc497220391)

[Лабораторная работа №№ 12-13 Работа с CASE–средствами кодирования программного обеспечения](#_Toc497220397) 81

[Лабораторная работа № 14 Работа с CASE–средствами тестирования программного обеспечения 90](#_Toc497220400)

[Приложение 1 Варианты заданий 102](#_Toc497220408)

Список использованных источников 109

### Лабораторная работа №1

### Тема: Создание диаграмм прецедентов (вариантов) использования

**Цель работы:** изучение среды программного инструмента моделирования StarUML, поддерживающего UML, и приобретение навыков по созданию диаграмм прецедентов (вариантов) использования.

**Теоретическая часть**

**Визуальное моделирование и UML**

**Визуальным моделированием** (visualmodeling)называется способпредставления идей и проблем реального мира с помощью моделей.

**Модель** –это абстракция,описывающая суть сложной проблемы илиструктуры без акцента на несущественных деталях, тем самым делая ее более понятной.

Разработка программного обеспечения - не исключение. При построении сложной системы строятся ее абстрактные визуальные модели. В настоящее время в области проектирования информационных систем успехом применяется визуальное моделирование с помощью унифицированного языка моделирования UML.

**Унифицированный язык моделирования** (UnifiedModelingLanguage, UML) является графическим языком для визуализации, специфицирования, конструирования и документирования систем, в которых большая роль принадлежит программному обеспечению.

С помощью UML можно детально описать систему, начиная разработку с концептуальной модели с ее бизнес-функциями и процессами, а также описать особенности реализации системы, такие как классы программного обеспечения системы, схему базы данных. Используя UML, мы также можем разрабатывать сложные системы быстро и качественно.

Как язык графического визуального моделирования UML имеет свою **нотацию** –принятые обозначения.Нотация обеспечивает семантику языка,является способом унификации обозначений визуального моделирования, обеспечивает всестороннее представление системы, которое сравнительно легко и свободно воспринимается человеком.

Моделирование с помощью UML осуществляется поэтапным построением ряда диаграмм, каждая из которых отражает какую-то часть или сторону системы либо ее замысла.

**Диаграмма** -это графическое представление множества элементов.Обычно диаграмма изображается в виде графа с вершинами (сущностями) и ребрами (отношениями). Диаграммы подчиняются нотации UML и изображаются в соответствии с ней.

Определены следующие виды диаграмм: вариантов использования (usecasediagram); классов (classdiagram); кооперации (collaborationdiagram);последовательности (sequencediagram); состояний (statechartdiagram); деятельности (activitydiagram); взаимодействия (interactiondiagrams); компонентов (componentdiagram); развертывания (deploymentdiagram).

Построения этих диаграмм достаточно для полного моделирования системы.

Особенности рабочего интерфейса программного комплекса StarUML

StarUML ориентирован на UML версии 1.4, поддерживает одиннадцать различных типов диаграмм, принятых в нотации UML 2.0, и представляет собой интегрированный инструментарий для проектирования архитектуры, анализа, моделирования и разработки программных систем.

Среда моделирования StarUML превосходно настраивается в соответствии с требованиями пользователя и имеет высокую степень расширяемости, особенно в области своих функциональных возможностей.

**Создание нового проекта в StarUML**

Основная структурная единица в StarUML – это проект. Проект сохраняется в одном файле в формате XML с расширением «.UML». Проект может содержать одну или несколько моделей и различные представления этих моделей (View) – визуальные выражения информации, содержащейся в моделях. Каждое представление модели содержит диаграммы – визуальные образы, отображающие определенные аспекты модели.

Новый проект будет автоматически создан при запуске программы StarUML. При этом вам будет предложено в диалоговом окне выбрать один из подходов (Approaches), поддерживаемых StarUML (см. рис. 1).

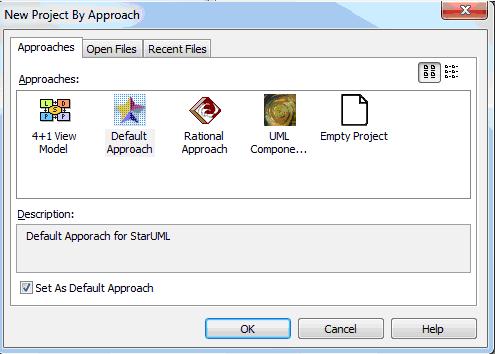


Рисунок 1. Выбор подхода

Существуют различные методологии моделирования информационных систем, компании-разработчики систем также могут разрабатывать свои методологии. Следовательно, на начальной стадии проектирования необходимо определить основные положения методологии или выбрать одну из уже существующих. Для того чтобы согласовать между собой различные элементы и этапы моделирования, StarUML предлагает концепцию подходов.

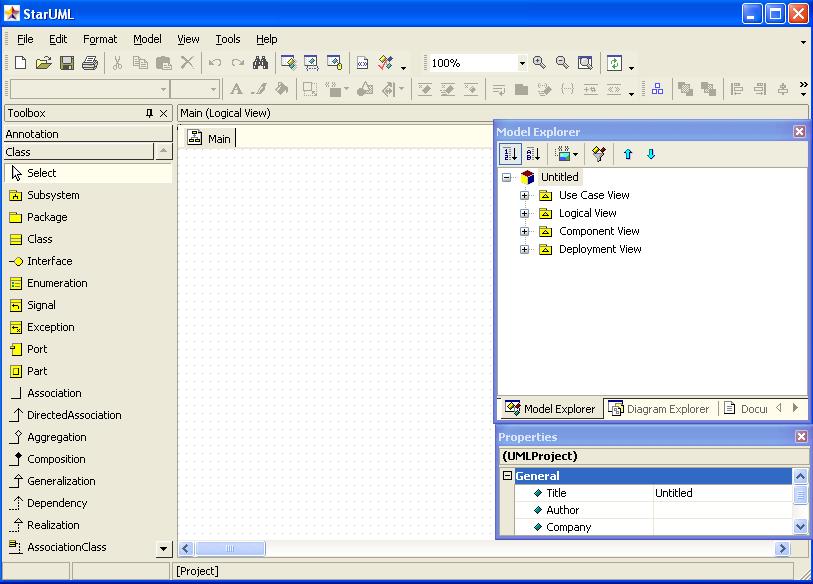
После выбора одного из предложенных подходов, появится основное окно программы (рис. 2).

Рисунок 2. Основное окно программы

Главное окно StarUML состоит из следующих компонентов:

* *главное меню* - находится наверху экрана. Большинство функции StarUML ™ доступны через главное меню;
* *инструментальные панели*, которые расположены ниже главного меню. Их кнопки дублируют часто используемые пункты меню;
* область браузера. Она расположена в верхнем правом углу экрана. Эта область содержит инструменты, облегчающие просмотр составляющих элементов проекта. Эта область включает [Навигатор модели] (*ModelExplorer*), который отображает модельные элементы в виде иерархической структуры, и [Навигатор диаграмм], (*DiagramExplorer*), который отображает диаграммы модели, сгруппированные по типам;
* область инспектора расположена в нижнем правом углу экрана. Эта область содержит инструменты, облегчающие редактирование детальной информации о модельных элементах. Эта область включает [Редактор свойств] (*Properties*), который позволяет редактировать свойства, [Документационный редактор] (Documentation), позволяющий водить детальные описания элементов, и [Редактор вложений] (Attachments), который позволяет присоединять к элементам дополнительные файлы и URL;
* иинформационная область расположена в нижней части экрана. Эта область содержит инструменты, отображающие различные вспомогательные данные, касающиеся приложения StarUML ™. Эта область включает [Окно Вывода] (*Output*), которое показывает регистрационную информацию, и [Окно Сообщений] (*Message*), которое отображает результаты поиска и проверки модели;
* область диаграммы расположена в центре экрана. Эта область содержит инструментарий, позволяющий редактировать и управлять диаграммами;
* палитра элементов расположена на левой стороне экрана. Палитра содержит инструменты для быстрого создания модельных элементов.

В верхней части окна расположено главное меню, кнопки быстрого доступа. Слева расположена панель элементов (Toolbox) с изображениями элементов диаграммы. Элементы соответствуют типу выбранной диаграммы. В центре находится рабочее поле диаграммы, на котором она может быть построена с использованием соответствующих элементов панели инструментов.

Справа находится **инспектор модели**, на котором можно найти вкладки навигатора модели ModelExploer, навигатора диаграмм DiagramExploer, окно редактора свойств Properties, окно документирования элементов модели Documentation и редактор вложений Attachments. Внешний вид инспектора модели с вкладками представлен ниже (рис. 3).

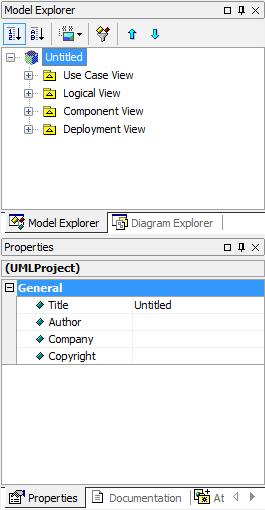


Рисунок 3. Инспектор модели

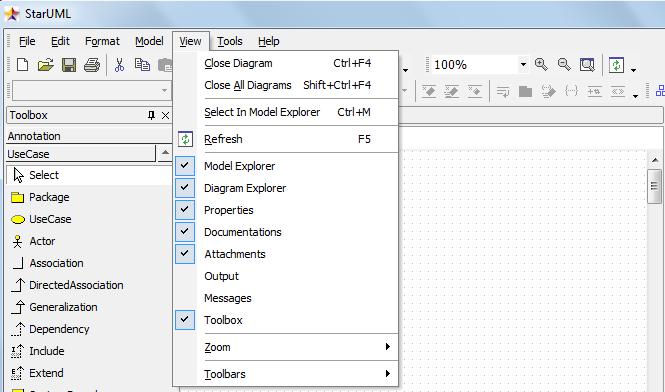


Рисунок 4. Пункт меню View. Управление видом инспектора модели

Управлять видом инспектора модели, панели элементов, закрывать и открывать редакторы инспектора можно с помощью пункта меню View (рис.4). Если рядом с пунктом меню стоит «галочка», этот элемент активен и его можно видеть в окне программы или открыть на доступных вкладках инспектора модели.

Иерархическая структура проекта отображается справа на навигаторе модели (ModelExploer). В зависимости от выбранного подхода на навигаторе модели будут отображены различные пакеты представлений модели. Каждый пакет представления будет содержать элементы моделей и диаграмм, которые мы создадим.

Если при создании нового проекта моделирования выбрать подход RationalApproach, то при таком подходе в навигаторе будут присутствовать четыре пакета представлений модели системы (см. рис. 5):

* UseCaseView – представление требований к системе, описывает, что система должна делать;
* LogicalView – логическое представление системы, описывает, как система должна быть построена;
* ComponentView – представление реализации, описывает зависимость между программными компонентами;
* DeploymentView – представление развертывания, описывает аппаратные элементы, устройства и программные компоненты.

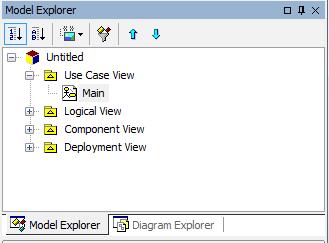


Рисунок 5. Навигатор модели

Сейчас каждое представление содержит одну диаграмму с именем Main. Если щелкнуть по ней два раза, то откроется рабочее поле этой диаграммы и соответствующая панель инструментов.

**Пример.** Если щелкнуть два раза по диаграммеMainпредставленияUseCaseView, то откроется рабочее этой диаграммы и ее панель элементов (рис. 6).

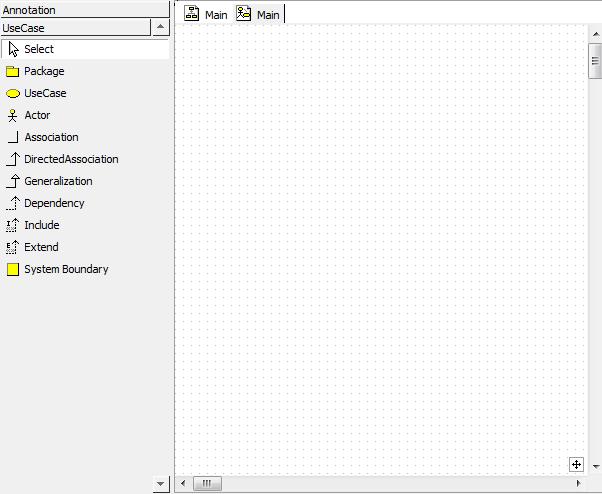


Рисунок 6. Рабочее поле диаграммы прецедентов Main и ее панель элементов

Ниже иерархии представлений отображаются свойства выделенного элемента модели или диаграммы (в данном случае свойства диаграммы Main, так как она выделена в навигаторе модели) (см. рис. 7).

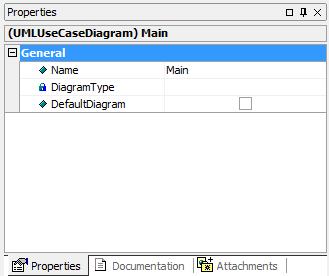


Рисунок 7. Редактор свойств

Назначение кнопок специальнойпанелиинструментов для диаграммыпрецендентовиспользования приведено в табл. 1.1.

|  |  |  |
| --- | --- | --- |
| Таблица 1.1 - Назначение кнопок специальной панели инструментов для диаграммы прецендентов использования | | |
| Графическое изображение | Всплывающая подсказка | Назначение кнопки |
| 3_1 | SelectionTool | Превращает изображение курсора в форму стрелки для последующего выделения элементов на диаграмме |
| 3_5 | Package | Добавляет на диаграмму пакет |
| 3_6 | UseCase | Добавляет на диаграмму прецендент использования |
| 3_7 | Actor | Добавляет на диаграмму актера |
|  | Association | Проводит семантическую ассоциацию между двумя сущностями |
| 3_8 | UnidirectionalAssociation | Добавляет на диаграмму направленную ассоциацию |
|  | Dependency | Соединяет два элемента отношением зависимости |
|  | Include | Соединят два прецедента отношением включения |
|  | Extend | Соединят два прецедента отношением расширения |
| 3_10 | Generalization | Добавляет на диаграмму отношение обобщения |
|  | SystemBoundary | Создаёт системную границу |

Порядок выполнения работы

Необходимо разработать модель прецедентов (вариантов) использования, состоящую из одной или нескольких диаграмм прецедентов использования, для разрабатываемой информационной системы.

**Постановка задачи (описание предметной области).** Магазин осуществляет продажу товаров клиенту путем оформления документов «Заказ». Директор магазина, принял решение автоматизировать документооборот продаж товара и пригласил для выполнения работ программиста. Поговорив с директором, в соответствие с концепцией жизненного цикла (ЖЦ) программы программист приступил к описанию бизнес процессов, сопровождающих продажу товара. Взяв за основу язык UML, он начал с построения контекстной диаграммы процессов- UseCasediagram. Диаграмма должна ответить на вопрос-«что должно делаться в системе и кто участник этих процессов». Окончательный вид диаграммы показан на рис.8

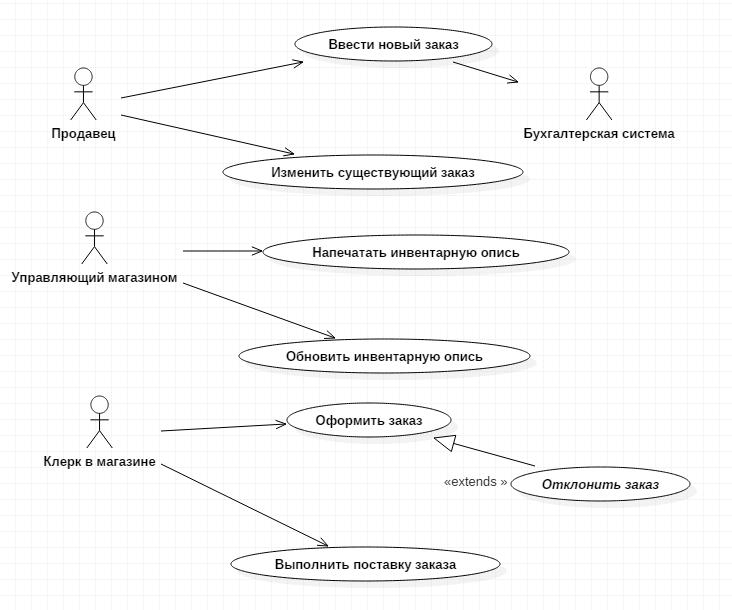


Рисунок 8 – Диаграмма вариантов использования задачи о заказе товара

Для этого выполните следующую последовательность действий:

1. Создайте новый проект и заполнитеегосвойства в областиProperties.
2. Внавигаторе моделей выберите <<useCaseModel>>, и с помощью контекстного меню выберите действие «AddDiagram» и создайте диаграмму прецедентов использования (UseCaseDiagram) для отображенияграницсистемы. Присвойте диаграмменаименование (область Properties) (рис.9).

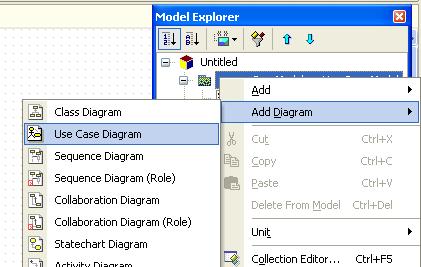


Рисунок 9 – Создание диаграммы прецедентов использования

1. С помощью окна «Toobox» и вкладки «UseCase» поместите на диаграмму новый вариант использования. Назовите его «Ввести новый заказ».
2. Повторив этапы 2 и 3, поместите на диаграмму остальные варианты использования:
   * Изменить существующий заказ;
   * Напечатать инвентарную опись;
   * Обновить инвентарную опись;
   * Оформить заказ;
   * Отклонить заказ;
   * Выполнить поставку заказа.
3. С помощью кнопки Actor (Действующее лицо) панели инструментов поместите на диаграмму новое действующее лицо.
4. Назовитеего «Продавец».
5. Повторив шаги 4 и 5, поместите на диаграмму остальных действующих лиц:

* Управляющиймагазином;
* Клеркмагазина;
* Бухгалтерскаясистема.

1. Создание абстрактного варианта использования (не требующего дальнейшей декомпозиции).

Щелкните мышью на варианте использования «Отклонить заказ» на диаграмме.

В окне «Properties» установите флажок isAbstract (Абстрактный), чтобы сделать этот вариант использования абстрактным (рис.10).

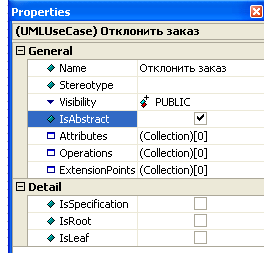


Рисунок 10 – Создание абстрактного варианта использования

1. Добавление ассоциаций. С помощью кнопкиDirectedAssociation (Однонаправленная ассоциация) в окне «Toolbox» нарисуйте ассоциацию между действующим лицом Продавец и вариантом использования «Ввести заказ».

Повторив шаг 1, поместите на диаграмму остальные ассоциации, согласно рис.8.

1. Добавление связи расширения. С помощью кнопкиGeneralization(Обобщение) в окне «Toolbox» нарисуйте связь между вариантом использования «Отклонить заказ» и вариантом использования «Оформить заказ». Стрелка должна быть направлена от первого варианта использования ко второму. Связь расширения означает, что вариант использования «Отклонить заказ» при необходимости дополняет функциональные возможности варианта использования «Оформитьзаказ».
2. Добавление описаний к вариантам использования. Выделите вариантиспользования «Ввести новый заказ». В окне документации введите следующее описание: «Этот вариант использования дает клиенту возможность ввести новый заказ в систему» (рис.11).

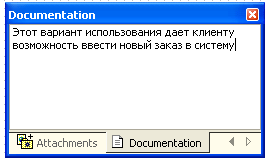


Рисунок 11 – Описание к вариантам использования

С помощью окна документации добавьте описания ко всем остальным вариантам использования.

1. Добавление описаний к действующему лицу. Выделите действующее лицо «Продавец». В окне документации введите следующее описание: «Продавец  это служащий, старающийся продать товар». С помощью окна документации добавьте описания к остальным действующим лицам.
2. Создайте еще одну или несколько диаграмм прецедентов использования, на которых отобразите все необходимые для системы прецеденты использования, актеров и все необходимые связи, например.



Рисунок 12 – Пример диаграммы описания границ системы

1. Диаграмма обязательно должна содержать включаемый и расширяющий прецеденты и соответствующие связи зависимости прецендентов (использования и расширения) между прецедентами. Для добавления связи использования необходимо нажать кнопку Include панели инструментов и направить связь от базового прецендента к включаемому. Для добавления связи расширения необходимо нажать кнопку Extend панели инструментов и направить связь к базовому прецеденту.

<<include>>

1. Вокнесвойствсвязирасширениянеобходимоуказать в текстовомвидеусловие (свойствоConditionсекцииGeneralобластиProperties ), при котором будет выполняться расширение.
2. В базовом прецеденте, имеющем расширение, необходимо создать точку расширения (свойствоExtenzionPointsсекцииGeneralобластиPropertiesпрецедента). Пример фрагмента диаграммы с точкой расширения приведен на рис. 13



Рисунок 13 - Пример фрагмента диаграммы с точкой расширения

На рис.14 отображен пример полной диаграммы прецедентов использования.

1. Создайте для одного из прецендентов использования файл, содержащий спецификацию потока событий (описание сценария его выполнения) и прикрепите его к преценденту (область Attachments).



Рисунок 14 – Диаграмма прецендентов использования для банкомата

1. Структура спецификации прецедента использования (usecasedocument) может варьироваться, однако типичное описание должно содержать следующую информацию:

* Краткое описание.
* Участвующие действующие лица.
* Предусловия, необходимые для инициирования прецедента использования.
* Детализированное описание потока событий, включающее в себя:
* основной поток событий, который можно разбить на несколько частей, чтобы показать подчиненные потоки событий (подчиненные потоки могут быть разделены затем на еще более мелкие потоки, с целью улуч­шить читабельность документа);
* альтернативные потоки для определения исключительных ситуаций.
* ■ Постусловия, определяющие состояние систем, по достижении которого прецедент использования завершается.

Пример спецификации потоков событий прецедента приведен в табл. 1.2.

Таблица 1.2 – Пример спецификации потоков событий

| Действия актеров | Отклик системы |
| --- | --- |
| Основной поток событий | |
| 1. Клиент вставляет кредитную карточку в устройство чтения банкоматаИсключение №1: Кредитная карточка недействительна | 2. Банкомат проверяет кредитную карточку  3. Банкомат предлагает ввести ПИН-код |
| 4. Клиент вводит персональный PIN-код  Исключение №2: Клиентвводит неверный ПИН-код | 5. Банкомат проверяет ПИН-код  6. Банкомат отображает опции меню |
| 7. Клиент выбирает снятие наличных со своего счета | 8. Система делает запрос в Банк и выясняет текущее состояние счета клиента  9. Банкомат предлагает ввести требуемую сумму |
| 10. Клиент вводит требуемую сумму  11. Банк проверяет введенную сумму  Исключение №3: Требуемая сумма превышает сумму на счете клиента | 12. Банкомат изменяет состояние счета клиента, выдает наличные и чек |
| 13. Клиент получает наличные и чек | 14. Банкомат предлагает клиенту забрать кредитную карточку |
| 15. Клиент получает свою кредитную карточку | 16. Банкомат отображает сообщение о готовности к работе |
| Исключение №1. Кредитная карточка недействительна или неверно вставлена | |
| Действия актера | Отклик системы |
|  | 1. Банкомат отображает информацию о неверно вставленной кредитной карточке  2. Банкомат возвращает клиенту его кредитную карточку |
| 3. Клиент получает свою кредитную карточку |  |
| Исключение №2. Клиент вводит неверный ПИН-код | |
|  | 1. Банкомат отображает информацию о неверном ПИН-коде |
| 2. Клиент вводит новый ПИН-код |  |
| Исключение №3. Требуемая сумма превышает сумму на счете клиента | |
|  | 1. Банкомат отображает информацию о превышении кредита |
| 2. Клиент вводит новую требуемую сумму |  |

1. После окончания сеанса работы над проектом выполненную работу необходимо сохранить в файле проекта с расширением .UMl.
2. Покажите разработанные диаграммы преподавателю.
3. Оформите отчет.
4. Сдайте выполненную и оформленную надлежащим образом работу преподавателю.

Содержание отчета

Индивидуальный отчет о выполненной лабораторной работе должен содержать:

1. цель работы;
2. наименование функционального комплекса задач или системы, для которых разработана диаграмма прецендентов использования;
3. разработанные диаграммы прецендентов использования;
4. спецификацию потоков событий для одного из прецендентов использования;
5. выводы о полученных знаниях и умениях.

Контрольные вопросы

1. Какие элементы может содержать диаграмма прецендентов использования?
2. Чем или кем могут быть представлены актеры (действующие лица) в системе?
3. В каких случаях используют отношения включения на диаграммах Прецендентов Использования?
4. В каких случаях используют отношения расширения на диаграммах прецендентов использования?
5. Как определяется направленность связи ассоциации между актером и прецендентом использования?
6. Что такое точка расширения прецедента?
7. С помощью чего описывается динамика прецендента использования.
8. На каком этапе разработки программной системы разрабатывается диаграмма прецедентов использования? Каково ее назначение?
9. Сколько диаграмм прецедентов использования необходимо разрабатывать для одной программной системы?

### Лабораторная работа №2

### Тема: Создание диаграмм последовательностей

**Цель работы:** изучение среды программного инструмента моделирования StarUML, поддерживающего UML, и приобретение навыков по созданию диаграмм последовательностей.

**Теоретическая часть**

**Диаграмма последовательности (sequencediagrams)**

Диаграмма последовательности отражает поток событий, происходящих в рамках варианта использования.

Все действующие лица показаны  в верхней  части  диаграммы. Стрелки  соответствуют сообщениям, передаваемым между действующим лицом и объектом или между объектами для выполнения требуемых функций.

На  диаграмме  последовательности  объект  изображается  в  виде прямоугольника,  от  которого  вниз  проведена  пунктирная  вертикальная линия.  Эта  линия  называется  линией  жизни (lifeline)  объекта.  Она представляет  собой  фрагмент  жизненного  цикла  объекта  в  процессе взаимодействия.

Каждое  сообщение  представляется  в  виде  стрелки  между  линиями жизни  двух  объектов. Сообщения  появляются  в  том  порядке,  как  они показаны  на странице  сверху  вниз.  Каждое  сообщение  помечается  как минимум  именем  сообщения. При  желании  можно  добавить  также аргументы и некоторую управляющую информацию. Можно показать самоделегирование (self-delegation) – сообщение, которое объект посылает самому себе, при этом стрелка сообщения указывает на ту же самую линию жизни.

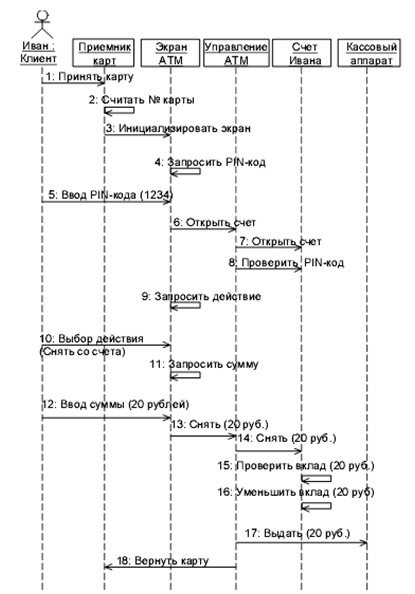


Рисунок 1 - Пример диаграммы последовательности

Порядок выполнения работы:

**Постановка задачи.** *Согласовав основные бизнес процессы с директором, прогаммист приступил к построению модели бизнес- процессов, что бы ответить на вопрос - «как это должно делаться в системе». Для начала он выбрал наиболее важный Вариант использования – «Ввод нового заказа» и построил для него диаграммы взаимодействия.*

*Диаграммы взаимодействия включают в себя два типа диаграмм-Последовательности и Кооперативную.*

1. **Создание диаграммы Последовательности.** В правом окне программы«ModelExplorer» на вкладке «Model» щелкнув правой кнопкой мыши, выберите действие «AddDiagram» и выберите тип диаграммы «SequenceDiagram».
2. Назовите новую диаграмму «Ввод заказа» во вкладке «Properties» в поле «Name».
3. Дважды щелкнув на этой диаграмме, откройте ее.

4.**Добавление на диаграмму действующего лица и объектов.** Перетащите действующее лицо «*Продавец»* из окна «ModelExplorer» из диаграммы использования на диаграмму «Ввод заказа»

1. Выберите объект «Lifeline» и поместите его на диаграмму.
2. Назовите объект *Выбор варианта заказа* (рис.1).
3. Повторив шаги 5 и 6, поместите на диаграмму объекты:
   * *Форма деталей заказа;*
   * *Заказ №1234.*

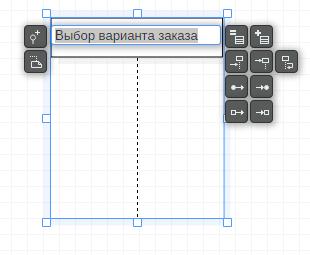


Рисунок 1 – Добавление объекта «*Выбор варианта заказа»*

1. **Добавление сообщений на диаграмму.** В окне«Toolbox»нажмите кнопку«Message» (Сообщение объекта).
2. Проведите мышью от линии жизни действующего лица «*Продавец»* к линии жизни объекта «*Выбор варианта заказа»*.
3. Введите его имя *Создать новый заказ*.
4. Повторив шаги 8 и 9, поместите на диаграмму сообщения:
   * *Открыть форму* -между*Выбор Варианта Заказа* и *Форма деталей Заказа;*
   * *Ввести номер заказа, заказчика и число заказываемых предметов* -между*Продавец*и *Форма Деталей Заказа;*
   * *Сохранить заказ* -между*Продавец*и *Форма Деталей Заказа;*
   * *Создать пустой заказ* -между*Форма Деталей Заказа* и *Заказ N1234;*
   * *Ввести номер заказа, заказчика и число заказываемых предметов* -между*Форма Деталей Заказа* и *Заказ N1234;*
   * *Сохранить заказ* -между*Форма Деталей Заказа* и *Заказ N1234.*

Завершен первый этап работы. Готовая диаграмма Последовательности представлена на рис. 2.

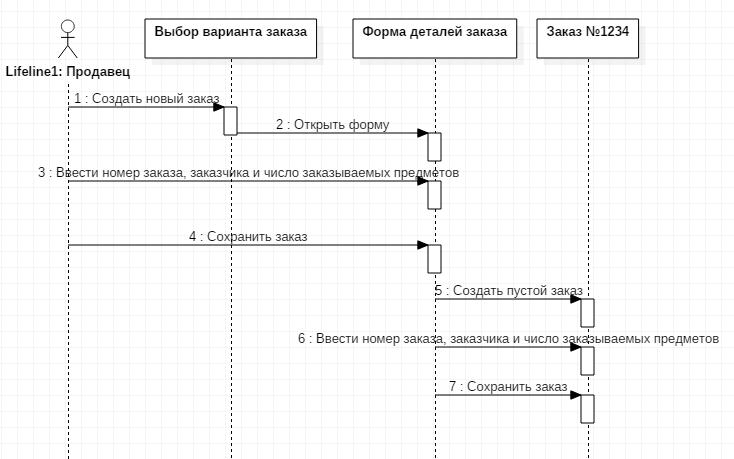


Рисунок 2– Диаграмма последовательности без управляющих элементов

Теперь нужно позаботиться об управляющих объектах и о взаимодействии с базой данных. Как видно из диаграммы, объект *Форма Деталей Заказа* имеет множество ответственностей, с которыми лучше всего мог бы справиться управляющий объект. Кроме того, новый заказ должен сохранять себя в базе данных сам. Вероятно, эту обязанность лучше было бы переложить на другой объект.

12. **Добавление на диаграмму дополнительных объектов.** Поместите новыйобъект между объектами *Форма Деталей Заказа* и *Заказ №1234*. Введите имя объекта «*Управляющий заказами»*.

* 1. Справа от*Заказ №1234* расположите другой новый объект под названием «*Управляющий транзакциями»*.
  2. **Назначение ответственностей объектам.** Выделите сообщение №5«*Создать пустой заказ»*.
  3. Нажав правой кнопкой мыши на сообщении, удалите его, применив действие «DeletefromModel».
  4. Повторите шаги 14 и 15 для удаления двух последних сообщений:
     + *Вести номер заказа, заказчика и число заказываемых предметов;*
     + *Сохранить заказ.*
  5. Поместите на диаграмму новое сообщение «*Сохранить заказ»*, расположив его под сообщением 4 между*Форма деталей заказа* и *Управляющий заказами*.
  6. Повторите шаги 16 – 17, добавив сообщения с шестого по девятое и назвав их:

*Создать новый заказ* между–*Управляющий заказами* и *Заказ №1234; Ввести номер заказа, заказчика и число заказываемых предметов* –между *Управляющий заказами* и *Заказ №1234; Сохранить заказ* –между *Управляющий заказами* и *Управляющий транзакциями; Информация о заказе* –между *Управляющий транзакциями* и *Заказ №1234.*

* 1. В окне «Toolbox» нажмите кнопку SelfMessage (Сообщение себе).
  2. Щелкните на линии жизни объекта *Управляющий транзакциями* ниже сообщения 9, добавив туда рефлексивное сообщение, назвав его «*Сохранитьинформацию о заказе в базе данных»*.
  3. **Соотнесение объектов с классами**.

1. Щелкните правой кнопкой мыши наобъекте *Выбор варианта заказа*.

2. В открывшемся меню выберите пункт OpenSpecification (Открыть спецификацию).

3. В раскрывающемся списке классов выберите пункт <New> (Создать). Появится окно спецификации классов.

1. В поле Name введите *Выбор заказа*.
2. Щелкните на кнопке ОК. Вы вернетесь в окно спецификации объекта.
3. В списке классов выберите класс *Выбор Заказа*.
4. Щелкните на кнопке ОК, чтобы вернуться к диаграмме. Теперь объект называется *Выбор варианта заказа:Выбор Заказа*
5. Для соотнесения остальных объектов с классами повторите шаги с 1 по 7:

* Класс *Детали заказа* соотнесите с объектом *Форма деталей заказа*
* Класс *Упр\_заказами* — с объектом *Управляющий заказами*
* Класс *Заказ* — с объектом *ЗаказN1234*
* Класс *Упр\_транзакциями* — с объектом *Управляющий транзакциями*

**Соотнесение сообщений с операциями**

1. Щелкните правой кнопкой мыши на сообщении 1: *Создать новый заказ.*

2. В открывшемся меню выберите пункт <newoperation> (создать операцию).

3. Появится окно спецификации операции.

1. В поле Name введите имя операции —*Создать*. Нажмите на кнопку ОК, чтобы закрыть окно спецификации операции и вернуться к диаграмме. Еще раз щелкните правой кнопкой мыши на сообщении 1.
2. В открывшемся меню выберите новую операцию*Создать()*.
3. Повторите шаги с 1 по 6, чтобы соотнести с операциями все остальные сообщения:

* Сообщение 2: *Открыть форму* соотнесите с операцией*Открыть()*
* Сообщение 3: *Ввести номер заказа,заказчика и число заказываемыхпредметов* —с операцией *Ввести номер заказа, заказчика и число заказываемых предметов()*
* Сообщение 4: *Сохранить заказ* — с операцией*Сохранить заказ()*
* Сообщение 5: *Сохранить заказ* — с операцией*Сохранить заказ()*
* Сообщение 6: *Создать пустой заказ* – с операцией*Создать пустой заказ()*
* Сообщение 7: *Ввести номер заказа,заказчика и число заказываемыхпредметов*-с одноименной операцией.
* Сообщение 8*Сохранить заказ* – с операцией *Сохранить заказ()*
* Сообщение 9 *Информация о заказе* – с одноименной операцией
* Сообщение 10*Сохранить информацию о заказе* с одноименной операцией. Диаграмма должна выглядеть, как на рис. 3.

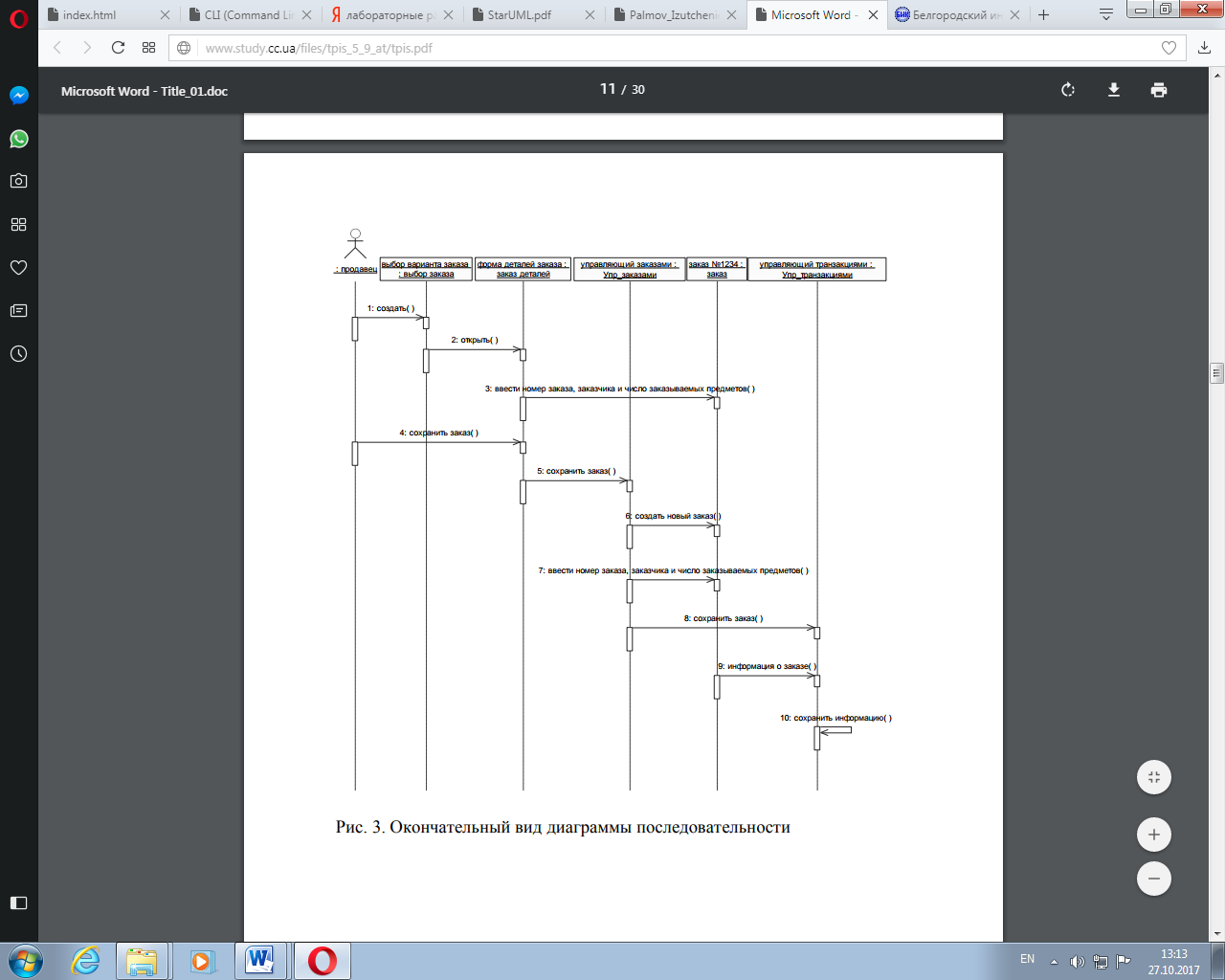


Рис. 3. Окончательный вид диаграммы последовательности

Контрольные вопросы

1.Что означает название UML и MDA?

2. Перечислите разные типы диаграмм в языке UML.

3.В каких областях науки и техники могут использоваться модели, созданные с помощью программы StarUML?

4.Перечислите основные концепции StarUML.

5.Что такое модель в программе StarUML?

6. Что такое представление, диаграмма, проект?

7. Какие элементы входят в структуру проекта?

8. Какое расширение имеют файлы проекта в StarUML?

9. Что такое фреймворк?

10. Какие профили используются в StarUML?

11. Какие базовые модули использует StarUML?

12. Перечислите основные типы диаграмм в программе StarUML.

### Лабораторная работа №3

### Тема: Создание кооперативной диаграммы

**Цель работы:** изучение среды программного инструмента моделирования StarUML, поддерживающего UML, приобретение навыков по созданию кооперативной диаграммы.

**Теоретическая часть**

**Диаграммы  взаимодействия (interactiondiagrams)**

Диаграммы взаимодействия (interactiondiagrams) 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.

*Сообщение (message)*  –  это  средство,  с  помощью  которого объект-отправитель запрашивает у объекта получателя выполнение одной из его операций.

*Информационное (informative)  сообщение* –  это  сообщение, снабжающее объект-получатель некоторой информацией для обновления его состояния.

*Сообщение-запрос (interrogative)* – это сообщение, запрашивающее выдачу некоторой информации об объекте-получателе.

*Императивное (imperative)  сообщение* –  это  сообщение, запрашивающее у объекта-получателя выполнение некоторых действий.

Существует  два  вида  диаграмм  взаимодействия:  диаграммы последовательности (sequencediagrams)  и  кооперативные  диаграммы (collaborationdiagrams).

**Диаграмма кооперации (collaborationdiagram)**

Диаграммы кооперации  отображают поток  событий  через  конкретный  сценарий варианта использования, упорядочены по времени,  а  кооперативные  диаграммы  больше  внимания  заостряют на связях  между  объектами.

Диаграмма кооперации – это альтернативный способ изображения сценария варианта использования. Этот тип диаграмм заостряет внимание на связях между объектами, отображая обмен данными в системе. А диаграммы последовательности отображают взаимодействие объектов во времени, поэтому ее следует читать сверху вниз и слева направо.

Диаграммы кооперации содержат все те же элементы, что и диаграммы последовательности: объекты, действующие лица, связи между ними и сообщения, которыми они обмениваются, но они уже не упорядочены во времени.

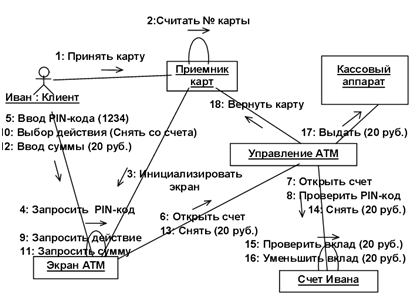


Рисунок1 - Пример диаграммы кооперации

Порядок выполнения работы:

**Постановка задачи.** Создать диаграмму кооперации. Конечный вид диаграммы представлен на рисунке 2.

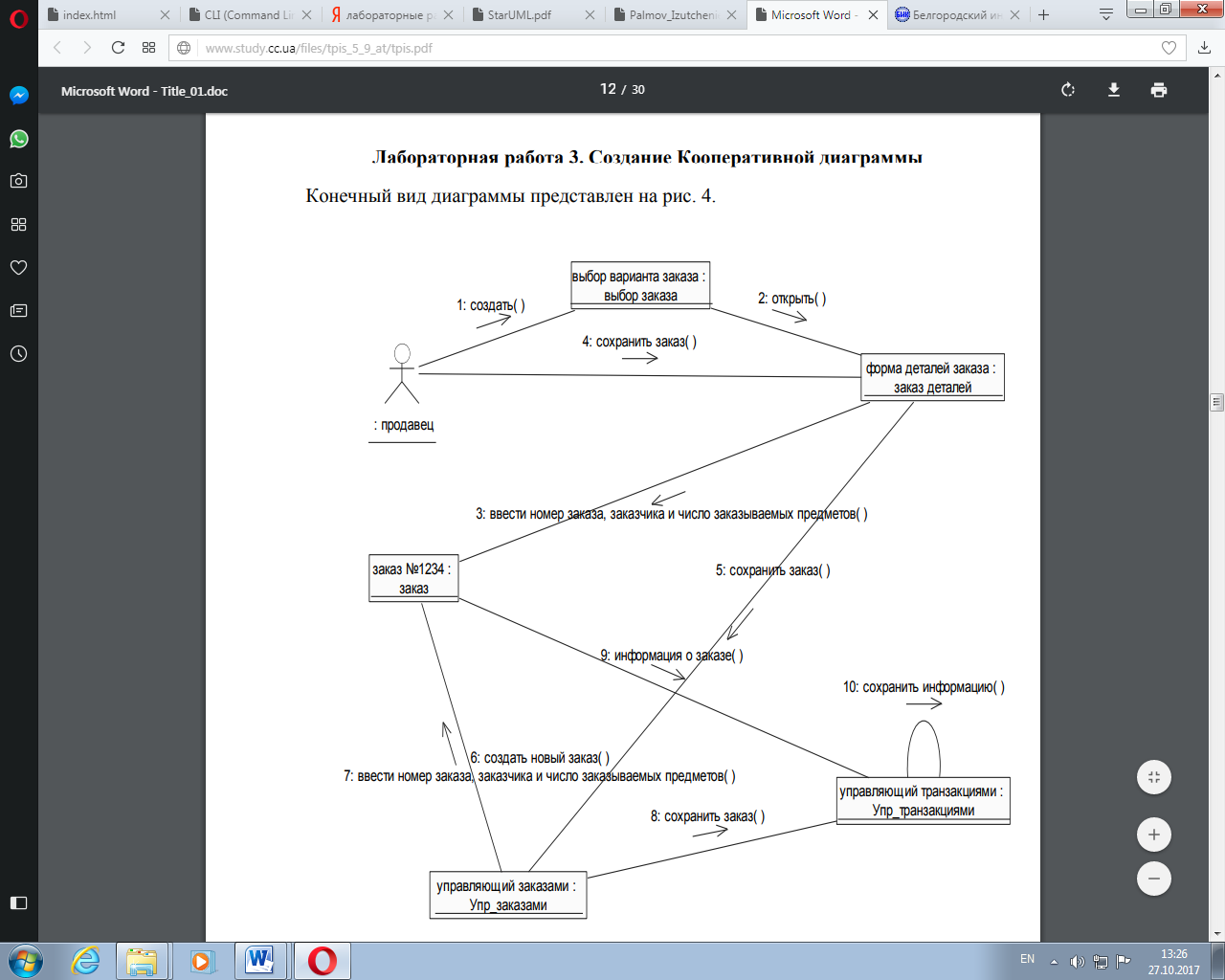


Рисунок 2 - Окончательный вид кооперативной диаграммы.

1. Для того чтобы добавить диаграмму кооперации в представление LogicalView, щелкните правой кнопкой мыши по папке содержащей диаграмму последовательности (если вы ее не переименовывали, то она носит имя CollaborationInstanceSet1), в контекстном меню выберите пункт AddDiagram, в списке выберите диаграмму кооперации Collaborationdiagram (рис. 3).

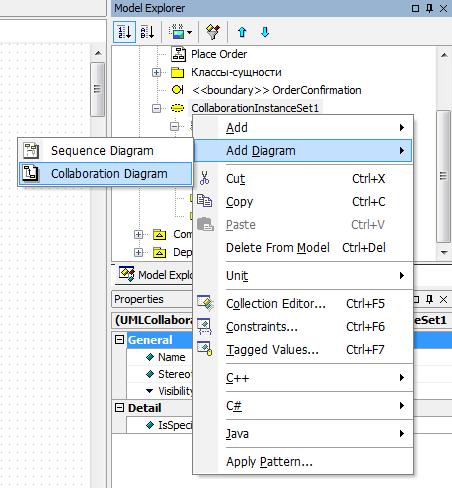


Рисунок 3- Добавление кооперативной диаграммы

1. Назовите эту диаграмму *Ввод заказа*.
2. Дважды щелкнув мышью на диаграмме, откройте ее.

**Добавление действующего лица и объектов на диаграмму**

1. Перетащите действующее лицо *Продавец* из браузера на диаграмму.
2. Нажмите кнопку Object (Объект) панели инструментов.
3. Щелкните мышью где-нибудь внутри диаграммы, чтобы поместить туда новый объект.
4. Назовите объект *Выбор варианта заказа*.
5. Повторив шаги 3 и 4, поместите на диаграмму объекты:

* *Форма деталей заказа*
* *Заказ №1234*

**Добавление сообщений на диаграмму**

1. На панели инструментов нажмите кнопку Link (Связь объекта).
2. Проведите мышью от действующего лица *Продавец* к объекту *Выборварианта заказа*.
3. Повторите шаги 1 и 2, соединив связями следующие объекты:

* Действующее лицо *Продавец* и объект *Форма деталей Заказа*
* Объект *Форма деталей Заказа* и объект *Выбор Варианта Заказа*
* Объект *Форма деталей Заказа* объект *ЗаказN1234*

1. На панели инструментов нажмите кнопку LinkMessage (Сообщение связи).
2. Щелкните мышью на связи между*Продавец* и *Форма деталей Заказа.*
3. Выделив сообщение, введите его имя —*Создать новый заказ*;
4. Повторив шаги с 4 по 6, поместите на диаграмму сообщения:

* *Открыть форму* —между*Выбор Варианта Заказа* и *Форма Деталей Заказа*.
* *Ввести номер заказа, заказчика и число заказываемых предметов* —между*Продавец*и *Форма Деталей Заказа*
* *Сохранить заказ* —между*Продавец*и *Форма деталей Заказа*
* *Создать пустой заказ* —между*Форма деталей Заказа* и *Заказ №1234*
* Ввести номер заказа, заказчика и число заказываемых предметов — между

*Форма деталей Заказа* и *Заказ №1234*

* Сохранить заказ — между*Форма деталей Заказа* и *Заказ №1234*

Теперь нужно поместить на диаграмму дополнительные элементы, а также рассмотреть ответственности объектов.

**Добавление на диаграмму дополнительных объектов**

1. Нажмите кнопку Object панели инструментов.
2. Щелкните мышью где-нибудь на диаграмме, чтобы поместить туда новый

объект.

1. Введите имя объекта — *Управляющий заказами*.
2. На панели инструментов нажмите кнопку Object.
3. Поместите на диаграмму еще один объект.
4. Введите его имя — *Управляющий транзакциями*.

**Назначение ответственностей объектам**

1. Выделите сообщение 5: *Создать пустой заказ*. Выделяйте слова, а не стрелку.
2. Нажав комбинацию клавиш CTRL+D, удалите это сообщение.
3. Повторите шаги 1 и 2 для удаления сообщений 6 и 7:

* *Ввести номер заказа, заказчика и число заказываемых предметов*
* *Сохранить заказ*

1. Выделите связь между объектами *Форма деталей Заказа* и *Заказ №1234*
2. Нажав комбинацию клавиш CTRL+D, удалите эту связь
3. На панели инструментов нажмите кнопку ObjectLink (Связь объекта).
4. Нарисуйте связь между*Форма деталей Заказа* и *Управляющий Заказами*.
5. На панели инструментов нажмите кнопку ObjectLink (Связь объекта).
6. Нарисуйте связь между*Управляющий Заказами* и *Заказ №1234*
7. На панели инструментов нажмите кнопку ObjectLink (Связь объекта).
8. Нарисуйте связь между*Заказ №1234* и *Управляющий Транзакцией*.
9. На панели инструментов нажмите кнопку ObjectLink (Связь объекта).
10. Нарисуйте связь между*Управляющий Заказами* и *УправляющийТранзакцией*.
11. На панели инструментов нажмите кнопку LinkMessage (Сообщение связи).
12. Щелкните мышью на связи между объектами *Форма деталей Заказа* и *Управляющий Заказами*,чтобы ввести новое сообщение.
13. Назовите это сообщение Сохранить заказ.
14. Повторите шаги 14 — 16, добавив сообщения с шестого по девятое, и назвав

их:

- *Создать новый заказ* — между*Управляющий Заказами* и *Заказ №1234* - *Ввести номер заказа,заказчика и число заказываемых предметов* — между *Управляющий Заказами* и *Заказ №1234* - *Сохранить заказ* — между *Управляющий Заказами* и *УправляющийТранзакцией* - *Информация о заказе* — между *Управляющий Транзакцией* и *Заказ №1234*

1. На панели инструментов нажмите кнопку LinktoSelf (Связь с собой).
2. Щелкнув на объекте *Управляющий Транзакцией*, добавьте к нему рефлексивное сообщение.
3. На панели инструментов нажмите кнопку LinkMessage (Сообщение связи).
4. Щелкните мышью на рефлексивной связи *Управляющий Транзакциями*, чтобы ввести туда сообщение.
5. Назовите новое*Сохранить информацию о заказе в базе данных*.

**Соотнесение объектов с классами (если классы были созданы при разработке описанной выше диаграммы Последовательности)**

1. Найдите в браузере класс *Выбор Заказа*.
2. Перетащите его на объект *Выбор варианта заказа* на диаграмме.
3. Повторите шаги 1 и 2 соотнеся остальные объекты и соответствующие им классы:

- Класс *заказ деталей* соотнесите с объектом *Форма деталей заказа* - Класс *Упр\_заказами* — с объектом *Управляющий Заказами*

- Класс *Заказ* — с объектом *Заказ №1234*

- Класс *Упр\_транзакциями* — с объектом *Управляющий транзакциями*

Соотнесение объектов с классами (если вы не создавали описанную выше диаграмму Последовательности)

1. Щелкните правой кнопкой мыши на объекте *Форма деталей Заказа*.
2. В открывшемся меню выберите пункт OpenSpecification (Открыть спецификацию).
3. В раскрывающемся списке классов выберите пункт <New> (Создать). Появится окно спецификации классов.
4. В поле имени введите *Выбор заказа*.
5. Щелкните на кнопке ОК. Вы вернетесь в окно спецификации объекта.
6. В списке классов выберите класс *Выбор заказа.*
7. Щелкните на кнопке OK, чтобы вернуться к диаграмме. Теперь объект называется *Выбор варианта заказа:Выбор Заказа*
8. Для соотнесения остальных объектов с классами повторите шаги с 1 по 7:

* Класс *Детали заказа* соотнесите с объектом *Форма деталей заказа*
* Класс *Упр\_заказами* — с объектом *Управляющий заказами*
* Класс *Заказ* — с объектом *ЗаказN1234*
* Класс *Упр\_транзакциями* — с объектом *Управляющий транзакциями*

**Соотнесение сообщений с операциями (если операции были созданы при разработке описанной выше диаграммы Последовательности)**

1. Щелкните правой кнопкой мыши на сообщении 1: Создать новый заказ.
2. В открывшемся меню выберите пункт OpenSpecification (Открыть спецификацию).
3. В раскрывающемся списке имен укажите имя операции — *Создать()*.
4. Нажмите на кнопку ОК.
5. Повторите шаги 1—4 для соотнесения с операциями остальных сообщений:

* Сообщение 2: *Открыть форму* соотнесите с операцией*Открыть()*
* Сообщение 3: *Ввести номер заказа,заказчика и число заказываемыхпредметов* —с операцией *Ввести номер заказа, заказчика и число заказываемых предметов()*
* Сообщение 4: *Сохранить заказ* — с операцией*Сохранить заказ()*
* Сообщение 5: *Сохранить заказ* — с операцией Сохранить заказ()
* Сообщение 6: *Создать пустой заказ* – с операцией Создать пустой заказ()
* Сообщение 7: *Ввести номер заказа,заказчика и число заказоваемых предметов* с одноименной операцией.
* Сообщение 8 Сохранить заказ – с операцией Сохранить заказ()
* Сообщение 9 *Информация о заказе* – с одноименной операцией
* Сообщение 10*Сохранить информацию о заказе* с одноименной операцией

**Соотнесение сообщений с операциями (если вы не создавали описанную выше диаграмму Последовательности)**

* 1. Щелкните правой кнопкой мыши на сообщении 1: *Создать новый заказ()*.
  2. В открывшемся меню выберите пункт <newoperation> (создать операцию). Появится окно спецификации операции.
  3. В поле имени введите имя операции —*Создать().*
  4. Нажмите на кнопку ОК, чтобы закрыть окно спецификации операции и вернуться к диаграмме.
  5. Еще раз щелкните правой кнопкой мыши на сообщении 1.
  6. В открывшемся меню выберите пункт OpenSpecification (Открыть спецификацию).
  7. В раскрывающемся списке Name (Имя) укажите имя новой операции.
  8. Нажмите на кнопку ОК.
  9. Повторите шаги 1—8, чтобы создать новые операции и соотнести с ними остальные сообщения:
* Сообщение 2: *Открыть форму* соотнесите с операцией*Открыть()*
* Сообщение 3: *Ввести номер заказа,заказчика и число заказываемыхпредметов -* с операцией *Ввести номер заказа, заказчика и число заказываемых предметов()*
* Сообщение 4: *Сохранить заказ* — с операцией*Сохранить заказ()*
* Сообщение 5: *Сохранить заказ* — с операцией*Сохранить заказ()*
* Сообщение 6: *Создать пустой заказ* – с операцией*Создать пустой заказ()*
* Сообщение 7: *Ввести номер заказа,заказчика и число заказоваемых предметов*- одноименной операцией
* Сообщение 8 Сохранить заказ – с операцией *Сохранить заказ()*
* Сообщение 9 *Информация о заказе* – с одноименной операцией
* Сообщение 10*Сохранить информацию о заказе* с одноименной операцией Ваша диаграмма должна выглядеть, как показано на рисунке 2.

Контрольные вопросы

1. Какие элементы может содержать диаграмма последовательности и диаграмма коопераций?
2. На каком этапе разработки программной системы разрабатывается диаграмма прецедентов взаимодействия? Каково ее назначение?
3. Сколько диаграмм взаимодействия необходимо разрабатывать для одной программной системы?

### Лабораторная работа №4

### Тема: Создание диаграммы состояний

**Цель работы:** изучение среды программного инструмента моделирования StarUML, поддерживающего UML, и приобретение навыков по созданию диаграммы состояний.

**Теоретическая часть**

**Диаграммы состояний**

Для представления взаимодействий объектов в системе используют диаграммы последовательности и кооперации. Прецеденты и сценарии описывают поведение системы, но если нам важна динамика класса, изменение состояний отдельного объекта, то для этой цели полезно построить диаграммы состояний.

***Диаграмма состояний*** показывает положение одиночного объекта,события и сообщения, которые вызывают переход из одного состояния в другое, и действия, являющиеся результатом смены состояния. Диаграмма состояний показывает объект с момента его создания и до его уничтожения.

Диаграмму состояний строят не для каждого класса в системе, а только для классов с динамическим поведением, которые отсылают и принимают много сообщений, изменяют свое состояние. Программный код из диаграмм состояний не генерируются, но они важны для понимания динамики поведения класса, дают возможность понять логику изменений перед кодированием.

Существует  много  форм  диаграмм  состояний,  незначительно отличающихся  друг  от  друга  семантикой.

На диаграмме имеются два специальных состояния – начальное (start) и  конечное (stop).  Начальное  состояние  выделено  черной  точкой,  оно соответствует  состоянию  объекта,  когда  он  только  что  был  создан. Конечное  состояние  обозначается  черной  точкой  в  белом  кружке,  оно соответствует  состоянию  объекта  непосредственно  перед  его уничтожением. На диаграмме состояний может быть одно и только одно начальное  состояние.  В  то  же  время,  может  быть  столько  конечных состояний,  сколько  вам  нужно,  или  их  может  не  быть  вообще.  Когда объект находится в каком-то конкретном состоянии, могут выполняться различные процессы.

Процессы,происходящие, когда объект  находится  в  определенном  состоянии,  называются  действиями (actions).

**Основные элементы диаграмм состояний**

Диаграмму состояний часто рассматривают в контексте конечного автомата. Тогда можно сказать, что ***диаграмма состояний(Statechartdiagram)*** показывает автомат,фокусируя внимание на потоке управления отсостояния к состоянию.

***Автомат (Statemachine)*** –это описание последовательностисостояний, через которые проходит объект на протяжении всего жизненного цикла, реагируя на события, - в том числе описание реакций на эти события.

***Состояние (State)*** –это ситуация в жизни объекта,на протяжениикоторой он удовлетворяет некоторому условию, осуществляет определенную деятельность или ожидает какого-то события.

Для поиска состояний класса можно просматривать атрибуты этого класса. Хорошим индикатором состояний является такой атрибут класса как «статус».

Диаграмма состояний изображается в виде графа с вершинами и ребрами.

Состояние на диаграмме изображается прямоугольником со скругленными вершинами. Под именем состояния могут размещаться действия (рис. 1).

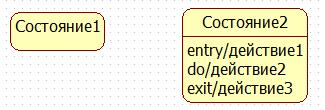


Рисунок 1 - Состояния

Кроме обычных состояний на диаграмме состояний могут размещаться псевдосостояния.

***Псевдосостояние (pseudo-state)* -**вершина в конечном автомате,которая имеет форму состояния, но не обладает поведением.

Примерами псевдосостояний, которые определены в языке UML, являются начальное и конечное состояния.

***Начальное состояние (startstate)* -**разновидность псевдосостояния,обозначающее начало выполнения процесса изменения состояний конечного автомата или нахождения моделируемого объекта в составном состоянии.

В этом состоянии находится объект по умолчанию в начальный момент времени. Оно служит для указания на диаграмме состояний графической области, от которой начинается процесс изменения состояний. Графически начальное состояние в языке UML обозначается в виде закрашенного кружка, из которого может только выходить стрелка-переход (рисунок 2)



Рисунок 2 - Начальное и конечное состояния

***Конечное состояние (finalstate)* -**разновидность псевдосостояния,обозначающее прекращение процесса изменения состояний конечного автомата (рис. 2).

Объект находится в конечном состоянии непосредственно перед уничтожением. Конечных состояний может быть несколько.

Находясь в каком-либо состоянии, объект может выполнять определенные действия, с состоянием можно связать такие действия как входное и выходное действия и внутренняя или просто деятельность.

С  состоянием  можно  связывать  данные  пяти  типов:  деятельность, входное  действие,  выходное  действие,  событие  и  история  состояния.

**Деятельность**

Деятельностью (activity)  называется  поведение,  реализуемое объектом, пока он находится в данном состоянии.  Деятельность –  это  прерываемое  поведение.  Оно  может выполняться  до  своего  завершения,  пока  объект  находится  в  данном состоянии,  или  может  быть  прервано  переходом  объекта  в  другое состояние. Деятельность изображают внутри самого состояния, ей должно предшествовать слово do (делать) и двоеточие.

**Входное действие**

Входным  действием (entryaction)  называется  поведение,  которое выполняется, когда объект переходит в данное состояние. Данное действие осуществляется не после того, как объект перешел в это состояние, а, скорее, как часть этого  перехода.  В  отличие  от  деятельности,  входное  действие рассматривается как непрерываемое. Входное  действие  также  показывают  внутри  состояния,  ему предшествует слово entry (вход) и двоеточие.

**Выходное действие**

Выходное  действие (exitaction)  подобно  входному.  Однако,  оно осуществляется  как  составная  часть  процесса  выхода  из  данного состояния. Оно  является  частью  процесса  такого перехода. Как и входное, выходное действие является непрерываемым.

Выходное действие изображают внутри состояния, ему предшествует слово exit (выход) и двоеточие.

Поведение объекта во время деятельности, при входных и выходных действиях может включать отправку события другому объекту. В  этом  случае  описанию  деятельности, входного  действия  или  выходного  действия  предшествует  знак  « ^ ».

Соответствующая строка на диаграмме выглядит как

*Do: ^Цель.Событие (Аргументы)*

Здесь  Цель –  это  объект,  получающий  событие,  Событие –  это посылаемое сообщение, а Аргументы являются параметрами посылаемого сообщения.

В общем случае действие имеет следующий формат:

<метка действия / выражение действия>

Деятельность  может  также  выполняться  в  результате  получения объектом  некоторого  события.   При  получении  некоторого  события  выполняется определенная деятельность.

**Срабатывание <перехода> (fire) - выполнение перехода из одного состояния в другое**.

На диаграмме переход изображается сплошной стрелкой. У перехода существует несколько спецификаций: событие, граничные условия, действия и посылаемые события.

<имя события>(<список параметров, разделенных запятыми>)[<сторожевое условие>]/<выражение действия>.

У  перехода  существует  несколько  спецификаций.  Они  включают события,  аргументы,  ограждающие  условия,  действия  и  посылаемые события.

**События**

Событие (event) – это то, что вызывает переход из одного состояния в другое. Событие размещают на диаграмме вдоль линии перехода.

На диаграмме для отображения события можно использовать как имя операции,  так  и  обычную  фразу.

Большинство переходов должны иметь события, так как именно они, прежде всего, заставляют переход осуществиться. Тем не менее, бывают и автоматические переходы, не имеющие событий. При этом объект сам перемещается из одного состояния в другое со скоростью, позволяющей осуществиться входным действиям, деятельности и выходным действиям.

**Ограждающие условия**

Ограждающие условия (guardconditions) определяют, когда переход может,  а  когда  не  может  осуществиться.  В противном случае переход не осуществится.

Ограждающие  условия  изображают  на  диаграмме  вдоль  линии перехода после имени события, заключая их в квадратные скобки.

Ограждающие  условия  задавать  необязательно.  Однако  если существует  несколько  автоматических  переходов  из  состояния, необходимо  определить  для них  взаимно  исключающие  ограждающие условия. Это поможет читателю диаграммы понять, какой путь перехода будет автоматически выбран.

**Действие**

После условий может указываться ***действие.*** Действием (action),  как  уже  говорилось,  является  непрерываемое поведение, осуществляющееся как часть перехода. Входные и выходные действия показывают внутри состояний, поскольку они определяют, что происходит, когда объект входит или выходит из него. Большую часть действий,  однако,  изображают  вдоль  линии  перехода,  так  как  они не должны осуществляться при входе или выходе из состояния.

Действие рисуют вдоль линии перехода после имени события, ему предшествует косая черта.

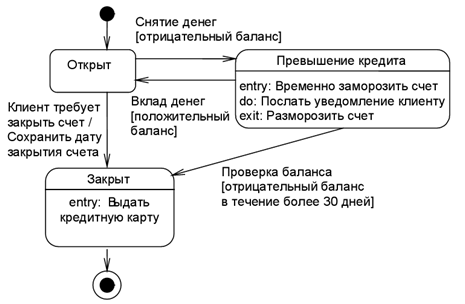


Рисунок3 - Пример диаграммы состояний

Диаграммы  состояний  не  надо  создавать  для  каждого  класса,  они применяются  только  в  сложных  случаях.  Если  объект  класса  может существовать  в  нескольких  состояниях  и  в  каждом  из  них  ведет  себя по-разному, для него может потребоваться такая диаграмма.

Событие  или  действие  могут  быть  поведением  внутри  объекта, а могут  представлять  собой  сообщение,  посылаемое  другому  объекту. Если событие  или  действие  посылается  другому  объекту,  перед  ним на диаграмме помещают знак « ^ ».

Порядок выполнения работы

## Постановка задачи

Проектируя класс Заказ, программист понял, что за поведением этого класса надо наблюдать. Многие требования к классу значительно изменялись при изменении состояния его экземпляра. Например, заказы, выполнение которых было приостановлено, вели себя не так, как выполненные заказы, а те, в свою очередь, не так, как отмененные заказы.

Чтобы убедиться, что проект удовлетворяет всем этим требованиям, программист со своей группой разработчиков создал диаграмму Состояний для класса Заказ. С помощью этой диаграммы разработчики смогли окончательно понять, как надо писать код для этого класса.

Постройте диаграмму Состояний для класса *Заказ*, показанную на рисунке 4



Рисунок4 - Диаграмма состояний для класса *Заказ*

**Создание диаграммы**

1. Найдите в браузере класс *Заказ*.
2. Щелкните на классе правой кнопкой мыши и в открывшемся меню укажите пункт AddDiagram>StatechartDiagram (Создать диаграмму состояний) (рисунок 5)

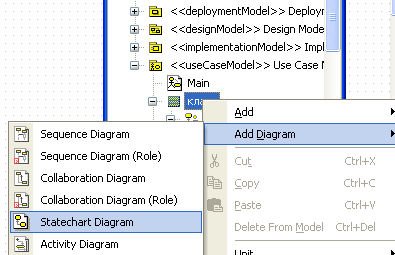
****

Рисунок 5 - Создание диаграммы состояний

**Добавление начального и конечного состояний**

1. Нажмите кнопку InitialState (Начальное состояние) панели инструментов.
2. Поместите это состояние на диаграмму.
3. Нажмите кнопкуFinalState (Конечное состояние)панели инструментов.

4.Поместите это состояние на диаграмму.

**Добавление суперсостояния**

1. Нажмите кнопку State (Состояние) панели инструментов.
2. Поместите это состояние на диаграмму.

**Добавление оставшихся состояний**

1. На панели инструментов нажмите кнопку State (Состояние).
2. Поместите состояние на диаграмму.
3. Назовите состояние *Отменен*.
4. На панели инструментов нажмите кнопку State(Состояние).
5. Поместите состояние на диаграмму.
6. Назовите состояние *Выполнен*.
7. На панели инструментов нажмите кнопку State(Состояние).
8. Поместите состояние на диаграмму внутрь суперсостояния.
9. Назовите состояние *Инициализация*.
10. На панели инструментов нажмите кнопку State (Состояние).
11. Поместите состояние на диаграмму внутрь суперсостояния.
12. Назовите состояние *Выполнение заказа приостановлено*.

**Описание состояний**

1. Дважды щелкните на состоянии Инициализация.

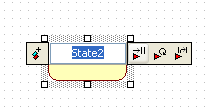


Рисунок 6 – Добавление списка внутренних действий

1. Выберите тип нового действия AddEntryAction.
2. Назовите его Сохранить дату заказа

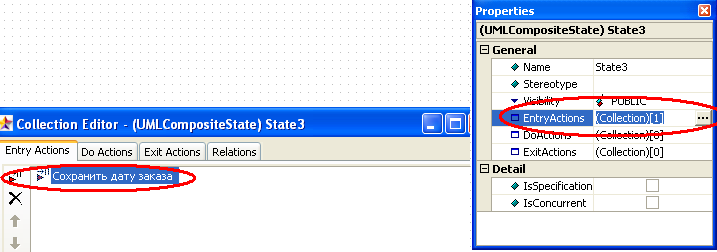


Рисунок 7 – Добавление действия EntryAction

1. Повторите этапы 1-3, добавив следующие действия:

* Собрать клиентскую информацию, DoAction
* Добавить к заказу новые графы, DoAction

1. Дважды щелкните на состоянии Отменен.
2. Повторите этапы 1 - 3, добавив действие Сохранить дату отмены, ExitAction
3. Дважды щелкните на состоянии Выполнен.
4. Повторите этапы 1 - 3, добавив действие Выписать счет, указать пункт DoAction.

**Добавление переходов**

1. Нажмите кнопку Transition (Переход) панели инструментов.
2. Щелкните мышью на начальном состоянии.

3. Проведите линию перехода к состоянию *Инициализация*.

4. Повторив шаги с первого по третий, создайте следующие переходы:

* От состояния Инициализация к состоянию *Выполнение заказа приостановлено*
* От состояния *Выполнение заказа приостановлено* к состоянию *Выполнен*
* От **суперсостояния** к состоянию *Отменен*
* От состояния *Отменен* к **конечному** состоянию
* От состояния *Выполнен* к **конечному** состоянию

1. На панели инструментов нажмите кнопку SelfTransition (Переход к себе).
2. Щелкните мышью на *состоянии Выполнение заказа приостановлено*

**Описание переходов**

1. Дважды щелкнув мышью на переходе от состояния *Инициализация* к состоянию*Выполнение заказа приостановлено*, откройте окно спецификации перехода (рисунок 8).
2. В поле Information введите фразу*Выполнить заказ*.

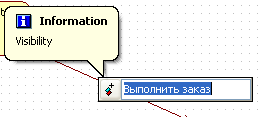
****

Рисунок 8**-** Окно спецификации перехода

1. Щелкнув на кнопке ОК, закройте окно спецификации.
2. Повторив шаги с первого по третий, добавьте событие*Отменить заказ* к переходу между суперсостоянием и состоянием *Отменен*.
3. Дважды щелкнув мышью на переходе от состояния *Выполнение заказаприостановлено* к состоянию *Выполнен*,откройте окно его спецификации.
4. В поле Information введите фразу*Добавить к заказу новую позицию*.
5. Перейдите на вкладку Properties (Свойства).
6. В поле GuardCondition (Сторожевое Условие) введите*Не осталосьнезаполненных позиций*.
7. Щелкнув на кнопке ОК, закройте окно спецификации.
8. Дважды щелкните мышью на рефлексивном переходе (SelfTransition) состояния *Выполнение заказа приостановлено*.
9. В поле Information введите фразу*Добавить к заказу новую позицию*.
10. Перейдите на вкладку Properties (Свойства).
11. В поле GuardCondition (Сторожевое Условие) введите*Остаютсянезаполненные позиции*.
12. Щелкнув на кнопке ОК, закройте окно спецификации.
13. Постройте диаграмму Состояний для объекта *Заказ*, показанную на рисунке 9.

**Пример.** В примере если покупатель получил заказ,то этособытие вызывает переход из состояния Укомплектован в состояние Выдан. Если же покупатель не получил свой заказ в течение двух недель, то заказ расформировывается, а деньги возвращаются покупателю на банковскую карту. Условие [Покупатель не забрал заказ в течение 2 недель] вызывает переход в состояние Расформирован при этом выполняется действие Вернуть деньги на карту.

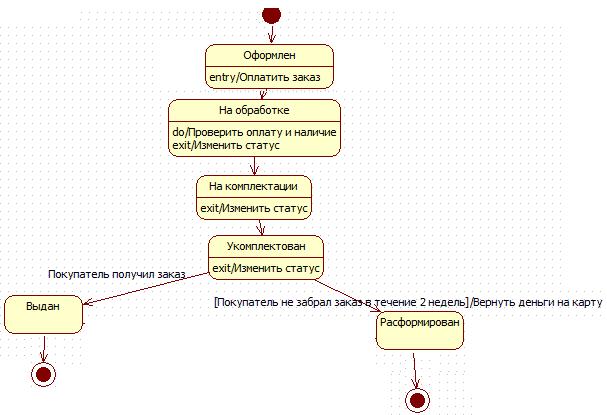


Рисунок 9 - Окончательная диаграмма состояний объекта Заказ

Контрольные вопросы

1. Для чего предназначены диаграммы состояний.
2. Основные элементы диаграммы состояний, их характеристики

### Лабораторная работа №5

### Тема: Создание диаграмм активности (деятельности)

**Цель работы:** изучение среды программного инструмента моделирования StarUML, поддерживающего UML, и приобретение навыков по созданию диаграмм активности (деятельности).

**Теоретическая часть**

#### Диаграмма активности (деятельности, activitydiagram)

На уроках информатики рисуют блок-схемы, чтобы наглядно изобразить алгоритм решения некоторой задачи. Действительно, моделируя поведение проектируемой системы, часто недостаточно изобразить процесс смены ее состояний, а нужно также раскрыть детали алгоритмической реализации операций, выполняемых системой. Для этой цели традиционно использовались блок-схемы или структурные схемы алгоритмов. В UML для этого существуют **диаграммы деятельности**, являющиеся частным случаем диаграмм состояний. Диаграммы деятельности удобно применять для визуализации алгоритмов, по которым работают операции классов.

**Алгоритм** - последовательность определенных действий или элементарных операций, выполнение которых приводит к получению желаемого результата.

Обозначения на диаграмме активности также напоминают те, которые встречаются на блок-схеме, хотя есть, и некоторые существенные отличия. С другой стороны, нотация диаграмм активности очень похожа на ту, которая используется в диаграммах состояний. Например, их можно увидить на диаграмме, которая представлена на рисунке 1

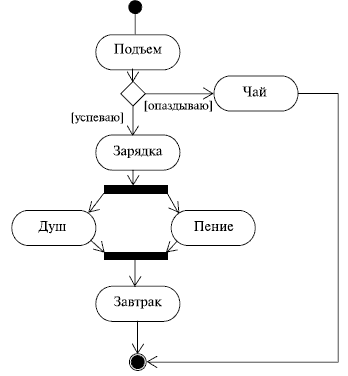


Рисунок 1 – Диаграмма деятельности

Обратите внимание на то, как изображено параллельное пение и принятие душа, - на обычной блок-схеме это было бы невозможно! А вот еще пример диаграммы, показывающей процесс оформления заказа в интернет-магазине (рисунок 2):

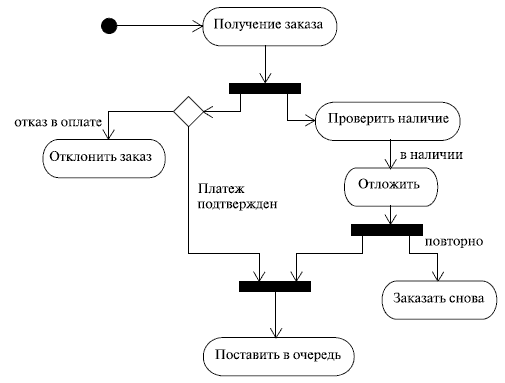


Рисунок 2 - Диаграмма деятельности оформления заказа в интернет-магазине

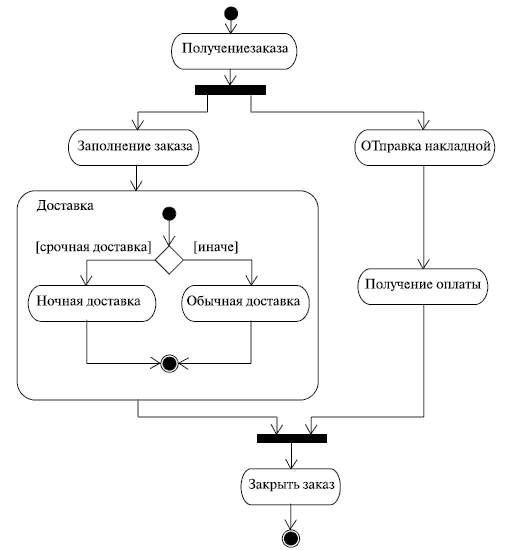


Рисунок 3 - Диаграмма деятельности доставки заказа

Activitydiagram — это специальная разновидность диаграммы состояний. В этом типе диаграмм большинство используемых знаков — это знаки активности, переходы между которыми вызваны завершением одних действий и началом других.

Главное отличие между Activity и Statechart в том, что в первом случае основное — действия, а во втором — статичное состояние. При этом Activitydiagram больше подходит для моделирования последовательности действий, а Statechart для моделирования дискретных состояний объекта.

Activity (активность) обозначает выполняемые задачи или выполнение определенных действий в течение жизни объекта.

StateTransition — переход из одного состояния в другое или по завершении выполнения определенного действия в начало другого.

Значок Synchronizations позволяет определить независимо выполняемые действия. При этом действия разделяются на несколько выполняемых независимо, и только по завершении всех действий объект продолжает работу. Этот значок представляет собой горизонтальную или вертикальную черту, обозначающую синхронизацию выполняемых работ.

Decision позволяет показать зависимость дальнейшей работы от внешних условий или решений. Этот значок аналогичен командам языка программирования IF или Case и может иметь больше двух выходов, но обычно используют выбор из двух переходов.

Swimlanes позволяет моделировать последовательность действий различных объектов и связи между ними. Swimlanes позволяет показать, кто выполняет те или иные роли в процессе. Для этого необходимо переместить соответствующие значки активности или состояний в зону определенного подразделения, отделенного от остальных Swimlanes.

**Построение диаграммы Активности для варианта использования «Выполнить поставку Заказа»**

**Постановка задачи:***Побеседовав с программистом, директор понял, что необходимо согласовать логику реализации еще одного варианта использования «Выполнить поставку заказа». Стало ясно, что здесь возможны несколько альтернативных потоков управления. Для таких ситуаций более удобно использовать не диаграммы взаимодействия, приспособленные для единственного потока, а диаграмму активности.*

**Описание варианта использования.**

При оформлении заказа проверяют каждую содержащуюся в нем позицию, чтобы убедиться в наличии соответствующих товаров на складе. После этого выписываются товары для реализации заказа. Во время выполнения этих процедур одновременно проверяется прохождение платежа. Если платеж прошел, и товары имеются на складе, то осуществляется их поставка. Если платеж прошел, но товары на складе отсутствуют, то заказ ставится в ожидание. Если платеж не прошел, то заказ аннулируется.

**Порядок выполнения работы:**

1. Найдите в браузере вариант использования «Выполнить поставку заказа»
2. Щелкните на классе правой кнопкой мыши и в открывшемся меню укажите пункт AddDiagram>ActivityDiagram (Создать диаграмму активности).
3. Назовите диаграмму «Выполнить поставку» и откройте ее двукратным щелчком мыши
4. На панели инструментов щелкните мышкой на элементе Swimlаne(Horizontal), затем на поле диаграммы.
5. Откройте окно Properties, измените поле Nameна*Клерк.*
6. Выполните заново пункты 4-5 и присвойте полю Name*Бухгалтерская система.*
7. Найдите в браузере сплошной черный кружок (начальное состояние). Перенесите его на дорожку *Клерк.*
8. Выберете из панели инструментов объект ActionState и поместите его на диаграмму в “дорожку” *Клерк*. Измените имя объекта на ”*Получить заказ*”.
9. Повторите предыдущий этап, создайте на «дорожке» *Клерк* 4 новых ActionState и присвойте им имена*Проверить позицию заказа,закрепить позицию за заказом,Поставить заказ в ожидание, Скомплектовать заказ*
10. Поместите на «дорожку» 2 новых объекта FinalState (конечное состояние). Одному из них измените поле Name на «*Выполнить поставку*»
11. На дорожку *Система* поместите новый объект Activity и присвойте полю Name “*Проверить платеж*”. На эту же дорожку поместите новый объект FinalState и измените его свойство Name на «*Отменить заказ*».
12. Поместить на «дорожку» Клерк 2 объекта Sinhronization (горизонтальная синхронизация). Присвойте полю Name спецификации одного объекта «*1*», другого- «*2*».
13. Поместить на «дорожку» *Клерк* объект Dicision (выбор), присвойте свойству Name «*Позиция имеется?*».
    1. Поместить на «дорожку» *Система* объект Dicision. Присвойте свойству Name «*Деньги поступили?*»
14. Щелкните мышкой на панели инструментов объекте- стрелке Transition (состояние перехода). Затем щелкните мышкой на диаграмме объекта начальное состояние. Удерживая кнопку мыши перенесите курсор на активность ”*Получитьзаказ*”и лишь затем отпустить курсор.В результатедваобъектабудутсоединеныстрелкой.
15. Выполните этап 14, соединив стрелкой объект Активность «*Получить заказ*» с объектом Sinhronization 1.
16. Соедините этими же стрелками объекты 1 и «Проверить платеж», 1 и «Проверить позицию заказа», «Проверить заказ» и «Деньги подступили?», «Деньги поступили?» и «Отменить заказ», «Проверить позицию заказа» и «Позиция имеется», «Позиция имеется» и «Закрепить позицию за заказом», «Деньги получены?» и 2, «Закрепить позицию за заказом» и 2, «Позиция имеется?» и «Поставить заказ в ожидание», 2 и «Скомплектовать заказ», «Скомплектовать заказ» и «Выполнить поставку», «Поставить заказ в ожидание» и объект Конечное состояние (без имени)
17. Присвойте некоторым стрелкам наименование (условие перехода).Для этого, установите курсор на стрелке, соединяющей «Деньги получены?» и «Отменить заказ». Двукратнымщелчкоммышиоткроетеокноинициализации, введите «Нет».
18. Выполните пункт 17 для стрелки, соединяющей «Деньги получены?» и 2 и присвойте значение «Да». Аналогично для стрелки соединяющей «Позиция имеется?» и «Закрепить позицию за заказом» присвоить «Да». Стрелке, соединяющей «Позиция имеется?» и «Поставить заказ в ожидание» - «Нет».
19. Добавить элементарные действия (Actions) к активности “Проверить позицию заказа», В поле Name ввести «Просмотреть спецификацию к заказу». Действие Do - «Найти новую позицию». Действие Exit (выход) - «Передать результаты поиска».
20. Путем перемещения объектов (установить курсор мыши- нажать- тащить-отпустить) привести диаграмму к виду, показанному на рисунке 6.

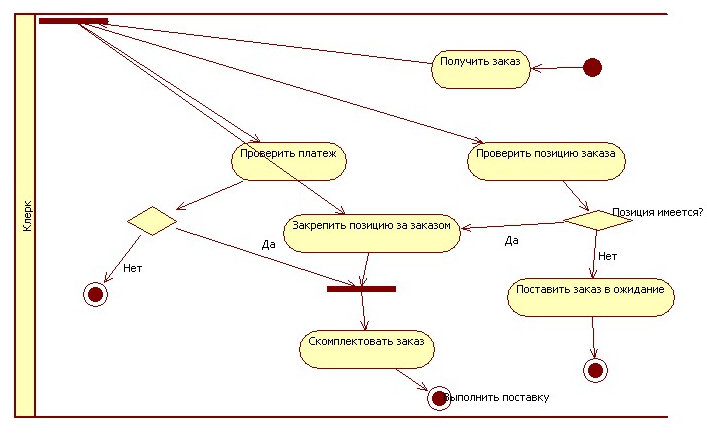
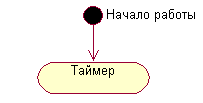


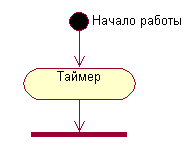
Рисунок 6 - Диаграмма активности для варианта использования «Выполнить поставку заказа»

**Задание 2.** Создайте диаграмму активности для процесса управления работой двигателя

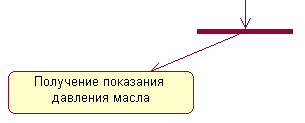
1. Создайте диаграмму активности, введите название диаграммы, например «Activity
2. С помощью значков  и  создайте начало и конец диаграммы соответственно. Назовите их «Начало работы» и «Конец работы». Для этого двойным щелчком левой кнопки мыши откройте спецификацию и в поле Name введите название. Нажмите ОК
3. С помощью значка  создайте активность и двойным щелчком левой кнопкой мыши по изображению активности откройте спецификацию и в поле Name введите название «Таймер». Нажмите ОК.



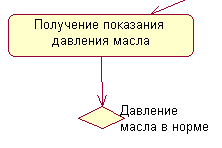
1. Создайте синхронизацию с помощью значка  на панели инструментов. Проведите переход значком  от изображения таймера до синхронизации



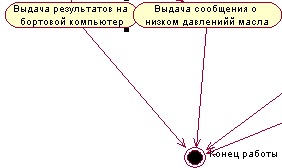
1. Значком  добавьте состояние и назовите его «Получение показаний давления масла», проведите состояние перехода от значка синхронизации к этому состоянию



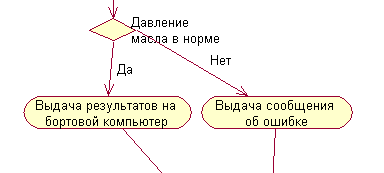
1. Создайте решение «Давление масла в норме» с помощью значка . Проведите состояние перехода от состояния «Получение показаний давления масла» к решению «Давлениемасла в норме»



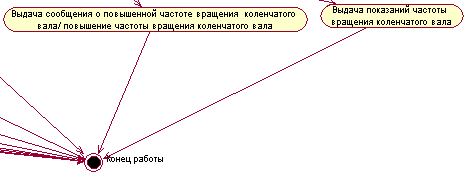
1. Сохраните диаграмму [File] – [Save] или нажав значок  на панели управления
2. Добавьте 2 активности и назовите их «Выдача результатов на бортовой компьютер» и «Выдача сообщения низком давлений масла». Соедините эти активности со значком «Конец работы»



1. Проведите 2 состояние перехода от решения «Давление масла в норме» до активностей созданных в пункте 11. Состояние перехода от решения «Давление масла в норме» до активности «Выдача результатов на бортовой компьютер» назовите «Да», а другое – «Нет»



1. Создайте состояние «Получение показаний напряжения аккумулятора» и проведите состояние перехода от значка синхронизации к этому состоянию
2. Создайте решение и назовите его «Напряжение аккумулятора в норме». Проведите состояние перехода от состояния «Получение показаний напряжения аккумулятора» к созданному решению
3. Добавьте 2 активности «Выдача сообщения о необходимости зарядки аккумулятора» и «Выдача показаний измерений». От решения «Напряжение аккумулятора в норме» проведите 2 состояния перехода к этим активностям. Состояние перехода от решения к активности «Выдача сообщения о необходимости зарядки аккумулятора» назовите «Нет», а другое – «Да»
4. Проведите состояния перехода от активностей «Выдача сообщения о необходимости зарядки аккумулятора» и «Выдача показаний на бортовой компьютер» к значку «Конец работы»
5. Создайте 3 состояния и назовите их «Получение показаний температуры двигателя», «Получение показаний расхода топлива» и «Получение показаний частоты вращения коленчатого вала». Проведите состояния перехода от значка синхронизации к этим состояниям
6. Создайте решение и назовите его «Температура двигателя в норме». Проведите состояние перехода от состояния «Получение показаний температура двигателя» к созданному решению
7. Добавьте 2 активности «Включение вентилятора охлаждения/ повышение частоты вращения коленчатого вала двигателя» и «Выдача показаний температуры двигателя». От решения «Температура двигателя в норме» проведите 2 состояния перехода к этим активностям. Состояние перехода от решения к активности «Включение вентилятора охлаждения/ повышение частоты вращения коленчатого вала двигателя» назовите «Нет», а другое – «Да»
8. Проведите состояния перехода от активностей «Включение вентилятора охлаждения/ повышение частоты вращения коленчатого вала двигателя» и «Выдача показаний температуры двигателя» к значку «Конец работы»
9. Создайте решение и назовите его «Расход топлива в норме». Проведите состояние перехода от состояния «Получение показаний расхода топлива» к созданному решению.
10. Добавьте 2 активности «Выдача сообщения о повышенном расходе топлива» и «Выдача показаний расхода топлива». От решения «Расход топлива в норме» проведите 2 состояния перехода к этим активностям. Состояние перехода от решения к активности «Выдача сообщения о повышенном расходе топлива» назовите «Нет», а другое – «Да»
11. Проведите состояния перехода от активностей «Выдача сообщения о повышенном расходе топлива» и «Выдача показаний расхода топлива» к значку «Конец работы».
12. Создайте решение и назовите его «Частота вращения коленчатого вала в норме». Проведите состояние перехода от состояния «Получение показаний частоты вращения коленчатого вала» к созданному решению.
13. Добавьте 2 активности «Выдача сообщения о повышенной частоте вращения коленчатого вала/ повышение частоты вращения коленчатого вала» и «Выдача показаний частоты вращения коленчатого вала». От решения «Частота вращения коленчатого вала в норме» проведите 2 состояния перехода к этим активностям. Состояние перехода от решения к активности «Выдача сообщения о повышенном расходе топлива» назовите «Нет», а другое – «Да»
14. Проведите состояния перехода от активностей «Выдача сообщения о повышенной частоте вращения коленчатого вала/ повышение частоты вращения коленчатого вала» и «Выдача показаний частоты вращения коленчатого вала» к значку «Конец работы». Диаграмма Activity для процесса управления двигателем готова.



**Задание 3.** Реализовать примеры из теоретической части

**Контрольные вопросы:**

1. Для чего может использоваться диаграмма Activity?
2. Для чего больше подходит Activitydiagram?
3. Какой инструмент который обозначает выполняемые задачи или выполнение определенных действий в течение жизни объекта?
4. Какой инструмент позволяет показать зависимость дальнейшей работы от внешних условий или решений?
5. Каково основное назначение (цели создания) диаграмм действий?
6. Почему для моделирования сложных систем необходимо использовать несколько видов диаграмм и моделей?
7. Каково назначение и цели создания языка UML?
8. Чем обусловлено многообразие UML-диаграмм?
9. Чем обусловлена необходимость построения моделей различных видов при описании предметной области?

### Лабораторная работа №№6-7

### Тема: Создание диаграмм классов

**Цель работы:** изучение среды программного инструмента моделирования StarUML, поддерживающего UML, и приобретение навыков по созданию диаграмм классов.

**Теоретическая часть**

**Диаграммы классов**

Диаграмма классов является частью логической модели системы и представляет статическую картину системы.

Для каждой системы строится не одна, а несколько диаграмм классов: возможно, что для каждого прецедента или сценария своя. На одних показывают подмножества классов, объединенные в пакеты, и отношения между ними, на других – отображают те же подмножества, но с атрибутами и операциями классов. Для представления системы разрабатывается столько диаграмм классов, сколько потребуется.

***Основные элементы диаграмм классов***

Дадим некоторые определения и опишем основные элементы нотации диаграмм классов.

***Объект*** –это некоторая сущность реального мира или концептуальная(абстрактная) сущность.

**Пример**.Примерами объектов могут служить дом №4по улицеСадовая, сотрудник фирмы Иван Петров, ваш компьютер. Или нечто абстрактное: химическая формула, торговый заказ номер 456789, банковский счет клиента Петра Иванова.

Объект имеет четко определенные границы и значение для системы и характеризуется состоянием, поведением и индивидуальностью.

***Состояние*** объекта–это одно из условий,в котором он можетнаходиться. Состояние обычно изменяется со временем и характеризуется набором свойств, которые называются атрибутами.

**Пример.** Покупатель определяется его именем,адресом,телефоном,датой рождения.

***Поведение*** определяет,как объект реагирует на запросы другихобъектов и что может делать сам объект. Поведение характеризуется операциями объекта.

**Пример.** Покупатель может добавить товар в корзину,просматриватькаталог, удалять товар из корзины.

***Индивидуальность*** означает,что каждый объект уникален,даже еслиего состояние идентично состоянию другого объекта.

**Пример.** Объекты Ирина Петрова и Анна Седова уникальны,хотякаждый из них является покупателем магазина и имеет одинаковые поведение и состояния.

Как правило, в системе существует множество объектов имеющих одинаковое поведение, принимающих одинаковые состояния. Например, сотрудники фирмы, которых может быть несколько десятков, и данные о которых содержатся в базе данных, имеют одинаковые атрибуты – фамилию, имя, отчество, дату рождения, должность и др. – с разными значениями этих атрибутов, а также могут иметь схожее поведение – подать заявление на отпуск или перевод в другое подразделение. Для группировки объектов используются классы.

***Класс*** –это описание группы объектов с общими свойствами(атрибутами), поведением (операциями), отношениями с другими объектами и семантикой.

Каждый класс является шаблоном для создания объекта. А каждый объект – это экземпляр класса. Важно помнить, что ***каждый объект может быть экземпляром только одного класса***

**Пример**.Применительно к магазинумы можем сгруппироватьсотрудников магазина, описав общий для них класс Сотрудник. Объект этого класса, например, Иван Петров, может включать в себя следующую информацию: имя, адрес, должность, размер заработной платы, кроме того этот объект может выйти в отпуск.

В нотации UML классы и объекты изображаются в виде прямоугольников (см. рис. 1). Прямоугольник класса всегда делится на три секции (раздела), имя класса помещается в первую секцию, каждое слово в названии класса принято писать с большой буквы. Во второй и третьей секциях могут указываться атрибуты и операции класса соответственно, эти секции могут быть пустыми. Названия классов выбираются в соответствии с понятиями предметной области. Это должно быть существительное или словосочетание в единственном числе, наиболее точно характеризующее предмет. Класс должен описывать только одну сущность.

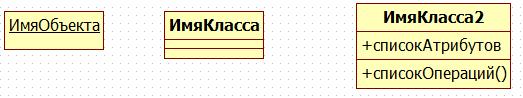


Рисунок 1- Изображение классов и объектов

Имя класса может быть простым, как это показано на рисунке 1, или составным (см. рис. 2). Составное имя класса состоит из самого имени класса и из имени пакета, которому принадлежит класс, разделенных двоеточием. Имя класса должно быть уникальным внутри пакета.

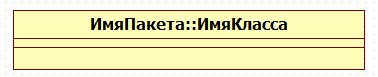


Рисунок 2 - Составное имя класса

Составное имя объекта также состоит из имени объекта и имени класса, разделенных двоеточием. Объект может быть анонимным, если неизвестно его настоящее имя. Тогда на диаграмме объект изображается с именем, которое состоит из двоеточия и имени класса, которому принадлежит объект. Если пока неизвестен класс, экземпляром которого является объект, то изображается имя объекта после которого идет двоеточие. Такой объект называется «сиротой» (см. рис. 3).

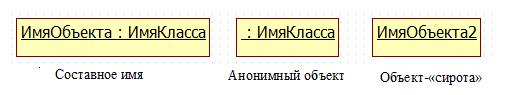


Рисунок 3 - Именование объектов

**Пример.** КлассСотрудники объект этого класса-некоторогосотрудника - можно изобразить так, как показано на рисунке 4.



Рисунок 4 - Класс и его объект

Мы дали объекту класса Сотрудник имя, совпадающее с именем класса.

***Выявление классов***

Выявление классов можно начать с изучения потока событий. Имена существительные в описании этого потока дадут понять, что может являться классом. В общем случае существительное может оказаться действующим лицом, классом, атрибутом класса или выражением, не являющимся ни действующим лицом, ни классом, ни атрибутом класса.

Если в ходе проектирования системы Вы уже построили диаграммы взаимодействия, перед тем, как приступать к построению диаграмм классов, то ищите на этих диаграммах похожие объекты. Например, у Вас может быть диаграмма последовательности, описывающая оформление заказа объектами Ивановым и Петровым. Обратите внимание на эти объекты: они имеют одинаковые свойства: имя, счет в банке и т.п. Значит, в системе должен появиться класс с именем Покупатель, который будет шаблоном объектов Иванов и Петров.

Некоторые возможные классы будут выявлены при рассмотрении трех стереотипов: сущность (entity), граница (boundary) и управление (control). Мы уже встречались со стереотипами отношений, когда говорили об отношениях на диаграммах прецедентов. Тот же принцип создания нового типа на основе уже существующего применим и для классов.

***Стереотип*** –это механизм,позволяющий категоризировать классы.Он используется для создания нового типа элемента, в данном случае нового типа класса.

Например, Вы хотите выделить все экранные формы в модели. Для этого нужно создать стереотип Form (Форма).

Стереотипы помогают лучше понять ответственности каждого класса в модели, категоризировать выполняемые ими функции. В UML для этого применяют три основных стандартных вида стереотипов классов: классы-сущности, граничные классы и управляющие классы.

***Класс-сущность*** содержит информацию,хранимую постоянно.Используется для моделирования данных и поведения с длинным жизненным циклом. Они могут представлять информацию о предметной области, а могут представлять элементы самой системы. Часто являясь абстракциями предметной области, они имеют наибольшее значение для пользователя, поэтому в их названиях применяются термины предметной области. Если существует проект базы данных, то можно обратиться к изучению названий таблиц, многие из них станут классами-сущностями. Обозначаются классы-сущности стереотипом <<entity>> либо специальной пиктограммой (рис. 5).

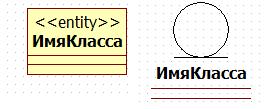


Рисунок 5 - Обозначение классов-сущностей

***Граничными классами*** называются классы,расположенные на границесистемы со всем остальным миром, и т.о. они обеспечивают взаимодействие между окружающей средой и внутренними элементами системы.

Для вычисления пограничных классов необходимо исследовать диаграммы вариантов использования. Для каждого взаимодействия между актером и прецедентом нужно создать хотя бы один граничный класс. Обратите внимание, что если два действующих лица инициируют один прецедент, то они могут применять один общий пограничный класс для взаимодействия с системой. Обозначаются граничные классы именем стереотипа <<boundary>> либо специальной пиктограммой (рис. 6).



Рисунок 6 - Обозначение граничных классов

***Управляющий класс*** отвечает за координацию действий другихклассов. Они служат для моделирования последовательного поведения одного или нескольких прецедентов и координации событий, реализующих заложенное в них поведение. Обозначаются управляющие классы именем стереотипа <<control>> либо специальной пиктограммой (рис. 7).

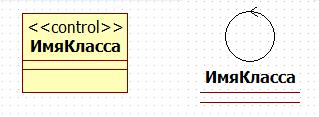


Рисунок 7 - Обозначение управляющих классов

Управляющие классы можно представить, как «исполняющие» прецедент, поэтому у каждого варианта использования обычно имеется один управляющий класс, контролирующий последовательность событий этого прецедента. Они обычно зависят от приложения.

Управляющий класс делегирует ответственности другим классам. Сам он может получать мало сообщений, но отсылать множество. Его называют классом-менеджером. Он запускает альтернативные потоки и знает, как поступить в случае ошибки. На начальном этапе проектирования управляющие классы создаются для каждой пары актер/прецедент, в дальнейшем они могут объединяться, разделяться или исключаться.

***Документирование классов***

После того, как класс создан, информацию о нем необходимо документировать. Заметим, что документация предназначена для описания предназначения класса, а не его структуры.

**Пример**.Если в нашей модели присутствует класс Сотрудник,тохорошим описанием для него будет:

*Сотрудник – это человек, работающий на фирме. Класс содержит информацию, необходимую для исполнения организацией своих обязанностей по отношению к сотруднику (начисление зарплаты, перевод на другую должность, увольнение и т.п.)*

Плохим описанием будет описание структуры класса, которая может быть и так описана с помощью атрибутов. Например, плохое описание класса Сотрудник:

*Имя, телефон, адрес, должность, зарплата.*

В StarUML документирование классов выполняется также как и описанное ранее документирование прецедентов. Нужно выделить класс, который вы хотите описать, открыть окно документирования Documentation на инспекторе модели и ввести описание класса.

**Пакеты в языке UML**

Если в нашей модели немного классов, то нам легко ими управлять, однако многие системы содержат большое количество классов, поэтому необходим механизм, позволяющий классы группировать и облегчающий их повторное использование. Таким механизмом в UML являются пакеты.

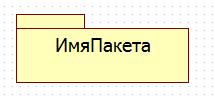
***Пакет (package)*** —общецелевой механизм для организацииразличных элементов модели в группы.

***Подпакет (subpackage)*** —пакет,который является составной частьюдругого пакета.

Пакет в логическом представлении модели – это объединение классов или других пакетов. С помощью объединения классов в пакеты мы можем получить представление о системе на более высоком уровне. Напротив, рассматривая пакет, мы получаем более детальное представление модели.

Объединять классы в пакеты можно как угодно, однако, существует несколько наиболее распространенных подходов.

* 1. можно группировать классы по стереотипам: классы-сущности, граничные и управляющие классы.
  2. группировка классов по их функциональности: например, пакет классов, отвечающих за безопасность системы или пакет классов Работа с сотрудниками и т.п.
  3. наконец, применяют комбинацию двух указанных методов.
* дальнейшем можно вкладывать пакеты друг в друга.

Чаще всего пакет на диаграмме изображается в виде папки с закладкой с именем пакета

Для того чтобы создать пакет на диаграмме, нужно открыть рабочее поле диаграммы, щелкнуть по элементу пакет Package на панели элементов слева, затем щелкнуть по рабочему полю диаграммы в том месте, где вы хотите поместить пакет. В окне редактора свойств можно задать новое имя пакета.

Чтобы разместить классы по пакетам, используют метод перетаскивания: на навигаторе модели нужно перетащить, удерживая левую кнопку мыши, классы в соответствующие пакеты на навигаторе модели.

**Постановка задачи:** *Необходимо сгруппировать в пакеты классы, созданные при выполнении предыдущих работ. Затем нужно будет построить несколько диаграмм Классов и показать на них классы и пакеты системы.*

**Порядок выполнения работы**

**Создание диаграммы Классов**

Объедините обнаруженные классы в пакеты. Создайте диаграмму Классов для отображения пакетов, диаграммы Классов, для представления классов в каждом пакете и диаграмму Классов для представления всех классов варианта использования "Ввести новый заказ".

**Этапы выполнения упражнения Создание пакетов**

1. В обозревателе моделей щелкните правой кнопкой мыши по разделу <<useCaseModel>> и выберите Add>Package (рисунок 8).

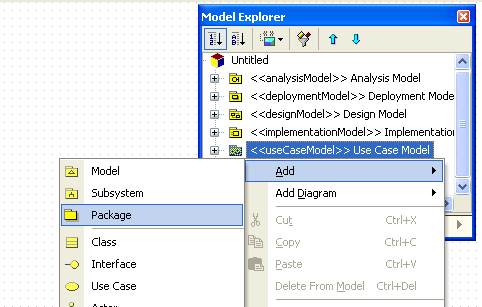


Рисунок 8 – Создание пакета

1. Назовите новый пакет Сущности.
2. Повторив шаги 1—2, создайте пакеты Границы и Управление.

**Создание Главной диаграммы Классов**

1. Внавигаторе моделей выберите с помощью контекстного меню создайте новую диаграмму классов **ClassDiagram**, откройте ее.
2. Перетащите пакет *Сущности* из браузера на диаграмму.
3. Перетащите пакеты *Границы* и *Управление* из браузера на диаграмму. Главная диаграмма Классов должна выглядеть, как показано на рис. 9

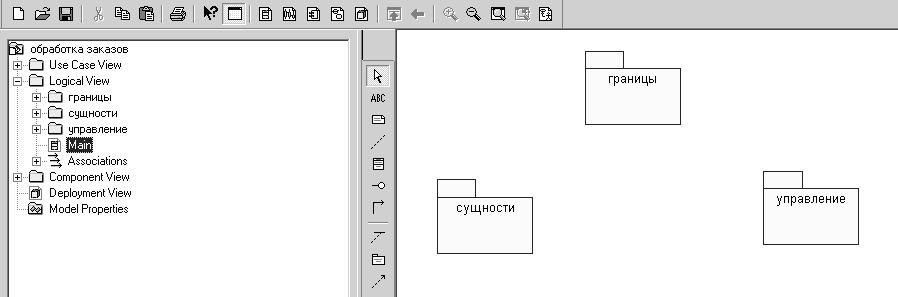


Рисунок 9 - Диаграмма классов в логическом представлении браузера.

**Создание диаграммы Классов для сценария "Ввести новый заказ" с отображением всех классов**

1. Внавигаторе моделей выберите с помощью контекстного меню создайте новую диаграмму классов **ClassDiagram**.
2. Назовите новую диаграмму Классов: Ввод нового заказа.
3. Дважды щелкнув мышью на этой диаграмме в браузере, откройте ее.
4. Перетащите из браузера все классы (*Выбор\_заказа,Заказ\_деталей,упр\_заказами, Заказ, Упр\_транзакциями*

**Объединение классов в пакеты**

1. В браузере перетащите класс *выбор\_заказа* на пакет Границы.
2. Перетащите класс *заказ\_деталей* на пакет Границы.
3. Перетащите классы *Упр\_заказами* и *Упр-транзакциями* на пакет Управление.
4. Перетащите класс *Заказ* на пакет Сущности.

Классы и пакеты в браузере показаны на рис. 10

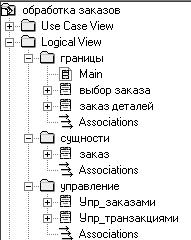


Рисунок 10 - Представление пакетов и классов Добавление диаграмм Классов к каждому пакету

1. В браузере щелкните правой кнопкой мыши на пакете *Границы*.
2. В открывшемся меню выберите пункт AddDiagram>ClassDiagram.
3. Введите имя новой диаграммы — Main (Главная).
4. Дважды щелкнув мышью на этой диаграмме, откройте ее.
5. Перетащите на нее из браузера классы *выбор\_заказа* и *заказ\_деталей*.
6. Закройте диаграмму.

В браузере щелкните правой кнопкой мыши на пакете *Сущности.*

1. В открывшемся меню выберите пункт AddDiagram>ClassDiagram.
2. Введите имя новой диаграммы — Main (Главная).
3. Дважды щелкнув мышью на этой диаграмме, откройте ее.
4. Перетащите на нее из браузера класс *Заказ*.
5. Закройте диаграмму
6. В браузере щелкните правой кнопкой мыши на пакете *Управление*
7. В открывшемся меню выберите пункт AddDiagram>ClassDiagram.
8. Введите имя новой диаграммы — Main (Главная).
9. Дважды щелкнув мышью на этой диаграмме, откройте ее.
10. Перетащите на нее из браузера классы *Упр\_заказами* и *Упр\_транзакциями*
11. Закройте диаграмму

**Уточнение методов и свойств классов**

В этом упражнении к описаниям операций будут добавлены детали, включая параметры и типы возвращаемых значений, и определены атрибуты классов

**Постановка проблемы**

*Для определения атрибутов классов был проанализирован поток событий. В результате к классу Заказ диаграммы Классов были добавлены атрибуты Номер заказа и Имя клиента. Так как в одном заказе можно указать большое количество товаров и у каждого из них имеются свои собственные данные и поведение, было решено моделировать товары как самостоятельные классы, а не как атрибуты класса Заказ.*

**Добавление атрибутов и операций**

Добавим атрибуты и операции к классам диаграммы Классов "Ввод нового заказа". При этом используем специфические для языка особенности. Установим параметры так, чтобы показывать все атрибуты, все операции и их сигнатуры. Применим нотацию UML.

***Этапы выполнения упражнения***

**Настройка**

1. В меню модели выберите пункт Тооls>Options (Инструменты >Параметры).
2. На вкладках General, Diagram, GeneralView выполните необходимые настройки:
3. Убедитесь, что флажок Showvisibility (Показать видимость) установлен.
4. Убедитесь, что флажок Showstereotyps (Показать стереотипы) установлен.
5. Убедитесь, что флажок Showoperationsignatures (Показать сигнатуры операций) установлен.
6. Убедитесь, что флажки Showallattributes (Показать все атрибуты) и Showalloperations (Показать все операции) установлены.
7. Убедитесь, что флажки Suppressattributes (Подавить атрибуты) и Suppressoperations (Подавить операции) сброшены.
8. Убедитесь, что флажок Visibilityasicons (Отображать пиктограммы) сброшен.

**Добавление нового класса**

1. Найдите в браузере диаграмму Классов варианта использования "Ввести новый заказ".
2. Дважды щелкнув мышью на диаграмме, откройте ее.
3. Нажмите кнопку С1аss панели инструментов.
4. Щелкните мышью внутри диаграммы, чтобы поместить туда новый класс.
5. Назовите его *Позиц\_заказа*.
6. Назначьте этому классу стереотип Entity (рисунок 11).

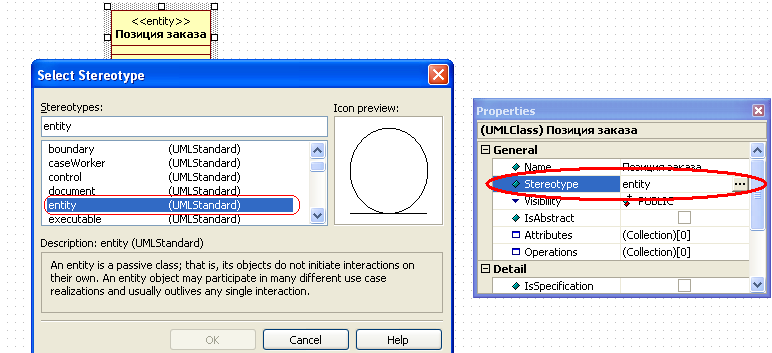
****

Рисунок 11 – Назначение классу Позиция заказа стереотипа Entity

1. В браузере перетащите класс в пакет *Сущности.*
2. Стереотипы классов можно отобразить с помощью пиктограмм. Для этого нужно выделить класс, щелкнуть по выделенной области правой кнопкой мыши, в контекстном меню выберите пункт Format, затем выберите пункт StereotypeDisplay, далее в списке выберите Iconic (рисунок 12)

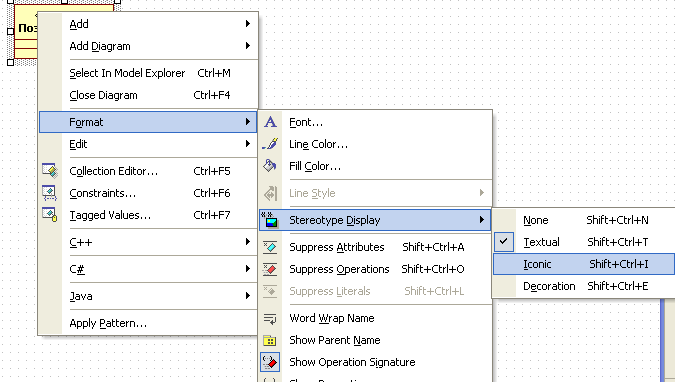
****

Рисунок 12 –Задание отображения стереотипов классов с помощью пиктограмм

Классы будут отображаться как пиктограммы (рисунок 13)

****

Рисунок 13 –Отображение классов с помощью пиктограмм стереотипов

**Добавление атрибутов**

1. Щелкните правой кнопкой мыши на классе *Заказ*.
2. В открывшемся меню выберите пункт NewAttribute (Создать атрибут),
3. Введите новый атрибут: **OrderNumber**:Integer
4. Нажмитеклавишу Enter
5. Введите следующий атрибут: **CustomerName**:String.
6. Повторив шаги 4 и 5, добавьте атрибуты: **OrderDate**:Date**OrderFillDate**:Date

Если тип атрибута не появляется в выпадающем списке, то введите его от руки и он далее будет появляться.

1. Щелкните правой кнопкой мыши на классе *Позиц\_заказа*.
2. В открывшемся меню выберите пункт NewAttribute (Создать атрибут).
3. Введите новый атрибут: ***ItemID*:Integer.**
4. Нажмите клавишу Enter.
5. Введите следующий атрибут: ***ItemDescription*:String.**

**Добавление операций к классу** *Позиц\_заказа*

1. Щелкните правой кнопкой мыши на классе *Позиц\_заказа*.
2. В открывшемся меню выберите пункт CollectionEditor и перейдите на вкладку Operations, то же самое можно выполнить выбрав свойство Operations в окне Properties (рисунок 14).
3. Введите новую операцию: *Создать()*

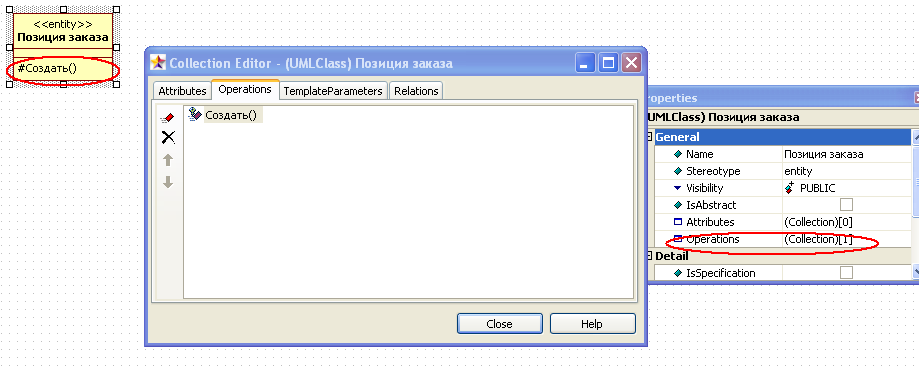


Рисунок 14 –Добавление операции к классу

Более быстрый способ создать атрибут или операцию – это щелкнуть два раза левой кнопкой мыши по классу (рисунок 15).



Рисунок 15. Быстрое создание атрибута

Нажав кнопку (для атрибута) или (для операции) вы получите атрибут или операцию соответственно.

Чтобы удалить атрибут или операцию щелкните по ней два раза левой кнопкой мыши и нажмите на значок  справа. Чтобы добавить еще один атрибут или операцию нажмите на значок  (рис. 16).

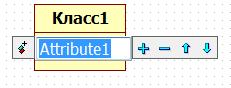


Рисунок 16. Быстрое добавление и удаление атрибута

1. Нажмите клавишу Enter.
2. Введите следующую **операцию:***Взять\_информацию()*
3. Нажмите клавишу Enter.
4. Введите операцию: *Дать\_информацию()*

**Подробное описание операций с помощью диаграммы Классов**

1. Щелкнув мышью на классе *Заказ*, выделите его.
2. Щелкните на этом классе еще раз, чтобы переместить курсор внутрь.
3. Отредактируйте операцию *Создать()*, чтобы она выглядела следующим образом:

**Создать() : Вооlеаn**

1. Отредактируйте операцию*Взять\_информацию*:

*Взять\_информацию*(OrderNum : Integer, Customer : String, OrderDate : Date,FillDate : Date) : Boolean

**5.** Отредактируйте операцию*Дать\_информацию*; *Дать\_информацию()***: String**

**Подробное описание операций с помощью браузера**

1. Найдите в браузере класс *Позиц\_заказа*.
2. Раскройте этот класс, щелкнув на значке "+" рядом с ним. В браузере появятся атрибуты и операции класса.
3. Дважды щелкнув мышью на операции *Дать\_информацию()* , откройте окно свойств:
4. В свойстве Type укажите тип String (рисунок 17).

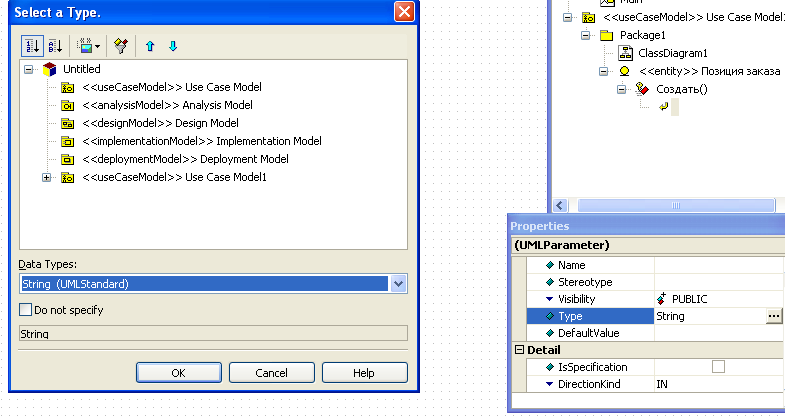
****

Рисунок 17. Определение типа атрибута

1. Щелкнув мышью на кнопке ОК, закройте окно спецификации операции.
2. Дважды щелкните в браузере на операции *Дать\_информацию* () класса *Позиц\_заказа*,чтобы открыть окно Properties.
3. В раскрывающемся списке DataTypes окна Select a type, которое вызывается свойством Types укажите Вооleаn.
4. Перейдите в обозревателей моделей.
5. Щелкните правой кнопкой мыши в области операции, чтобы добавить туда новый параметр.
6. В открывшемся меню выберите пункт Add>Parametr. Будет добавлен параметр под названием Parametr1 (рисунок 18).

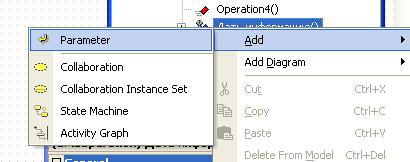
****

Рисунок 18 – Добавление параметра операции

1. В окне Properties измените имя аргумента на ID.
2. Щелкните на свойстве Туре (Тип). В раскрывающемся списке типов выберите Integer (Если этого либо иного необходимого типа не будет- введите его вручную).
3. Щелкните на свойстве DefaultValue (По умолчанию), чтобы добавить значение аргумента по умолчанию. Введите число 0.
4. Дважды щелкните на операции *Создать()* класса *Позиц\_заказа*, чтобы открыть окно ее свойств.
5. В раскрывающемся списке DataTypes окна Select a type, которое вызывается свойством Types укажите Вооleаn.
6. Нажав на кнопку ОК, закройте окно.

**Подробное описание операций**

1. Используя браузер или диаграмму Классов, введите следующие сигнатуры операций класса *Заказ\_деталей*: *Открыть()*:Boolean, *Сохранить заказ()* : Boolean
2. Используя браузер или диаграмму Классов, введите сигнатуру операций класса *Выбор\_заказа*: *Создать()* :Вооlеаn
3. Используя браузер или диаграмму Классов, введите сигнатуру операций класса *Упр\_заказами*: *Сохранить заказ*(OrderID :Integer) :Вооlеаn
4. Используя браузер или диаграмму Классов, введите сигнатуры операций класса *Упр\_транзакциями*: *Сохранить заказ* (OrderID :Integer) :Boolean, *Сохранить информацию*() :Integer

**Описание связей между классами**

В этом упражнении определяются связи между классами, участвующими в варианте использования "Ввести новый заказ".

**Постановка задачи**

*Чтобы найти связи, были просмотрены диаграммы Последовательности. Все взаимодействующие там классы нуждались в определении соответствующих связей на диаграммах Классов. После обнаружения связи были добавлены на диаграммы классов.*

*Добавление связей*

Добавим связи к классам, принимающим участие в варианте использования "Ввести новый заказ".

***Этапы выполнения упражнения***

**Добавление ассоциаций**

1. Найдите в браузере диаграмму Классов "Ввод нового заказа",
2. Дважды щелкнув на диаграмме, откройте ее.
3. Нажмите кнопку DirectedAssociation панели инструментов.
4. Проведите ассоциацию от класса *выбор\_заказа* к классу *заказ\_деталей*.
5. Повторите шаги 1 и 2, создав ассоциации:

* От класса *заказ\_деталей* к классу *упр\_заказами*
* От класса *упр\_заказами* к классу *Заказ*
* От класса *упр\_заказами* к классу *упр\_транзакциями*
* От класса *упр\_транзакциями* к классу Заказ
* От класса *упр\_транзакциями* к классу *Позиц\_заказа*
* От класса *Заказ* к классу *Позиц\_заказа*

1. Щелкните правой кнопкой мыши на однонаправленной ассоциации между классами *выбор\_заказа* и *заказ\_деталей* класса *выбор\_заказа*.
2. В открывшемся меню выберите пункт Multiplicity> (Множественность >- Нуль или один). На рисунке 19 наглядно показано это действие.

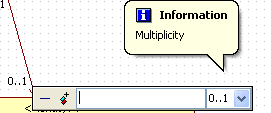
****

Рисунок 19 – Задание множественности

Задать это значение можно также пользуясь свойствами End.Multiplicity, End2.Multiplicity окна Properties (рисунок 20)

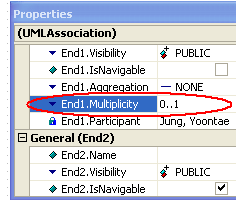
****

Рисунок 20 **-** Задание свойства множественности в окне Properties

1. Щелкните правой кнопкой мыши на другом конце однонаправленной ассоциации.
2. В открывшемся меню выберите пункт Multiplicity> (Множественность *>* Нуль или один),
3. Повторите шаги 4—7, добавив на диаграмму значения множественности для остальных ассоциаций, как показано на рис. 21

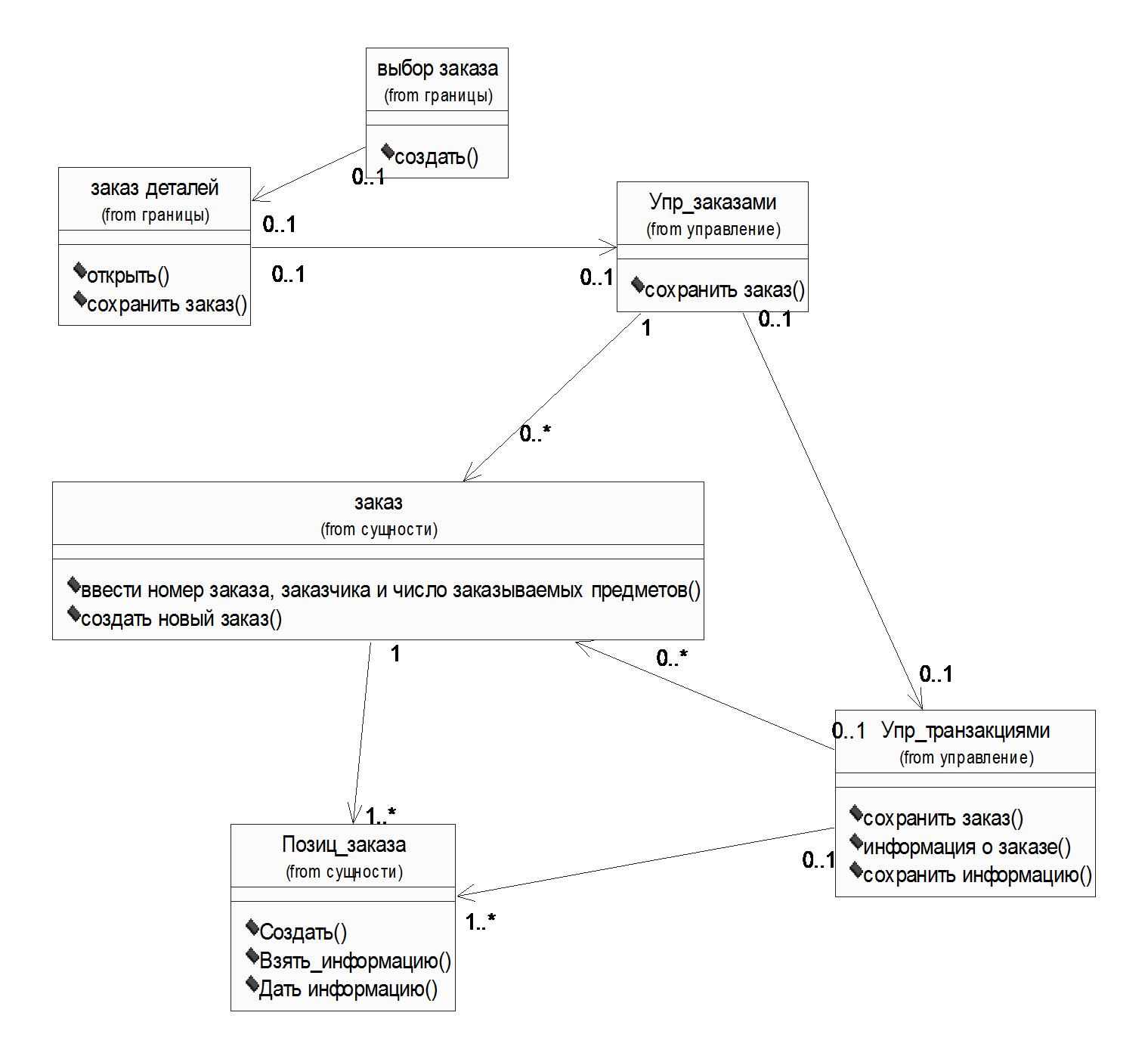


Рисунок 21 - Ассоциации сценария "Ввести новый заказ"

**Задание+:** Диаграмму классов сценарияОформление заказаотобразить с помощью пиктограмм стереотипов

Содержание отчета

Индивидуальный отчет о выполненной лабораторной работе должен содержать:

1. цель работы;
2. наименование функционального комплекса задач или задачи, для которых разработана диаграмма прецедентов использования;
3. разработанную диаграмму классов;
4. выводы о полученных знаниях и умениях.

Контрольные вопросы

1. Какие элементы может содержать диаграмма классов?
2. Какие свойства спецификации имеет класс?
3. Что называют сообщением в UML?
4. Какого типа отношения существуют между классами?
5. В каком случае устанавливается связь агрегации? Как определяется ее направленность?
6. Чем отличается отношение композиции от отношения агрегации?
7. В каком случае устанавливается связь обобщения? Как определяется ее направленность?
8. Каким образом может быть использована диаграмма классов?
9. В каком случае необходимо именовать отношение ассоциации или ее полюсов?
10. Прообразом чего в программном коде является операция класса?
11. Что такое интерфейс? В каких случаях в диаграмму классов включают интерфейсы?
12. Как определить, является ли класс классом сущности?

### Лабораторная работа № 8-9

### Тема: Работа с инструментальными средствами, поддерживающими методологию объектно-ориентированного моделирования

# Цель работы: Ознакомление с основными элементами определения, представления, проектирования и моделирования программных систем с помощью языка UML.

**Теоретическая часть**

Существует множество технологий и инструментальных средств, с помощью которых можно реализовать в некотором смысле оптимальный проект ИС, начиная с этапа анализа и заканчивая созданием программного кода системы. В большинстве случаев эти технологии предъявляют весьма жесткие требования к процессу разработки и используемым ресурсам, а попытки трансформировать их под конкретные проекты оказываются безуспешными. Эти технологии представлены CASE-средствами верхнего уровня или CASE-средствами полного жизненного цикла (upper CASE tools или fulllife-cycle CASE tools). Они не позволяют оптимизировать деятельность на уровне отдельных элементов проекта, и, как следствие, многие разработчики перешли на так называемые CASE-средства нижнего уровня (lower CASE tools). Однако они столкнулись с новой проблемой — проблемой организации взаимодействия между различными командами, реализующими проект.

**Унифицированный язык объектно-ориентированного моделирования UnifiedModelingLanguage (UML)** явился средством достижения компромисса между этими подходами. Существует достаточное количество инструментальных средств, поддерживающих с помощью *UML* жизненный цикл информационных систем, и, одновременно, *UML* является достаточно гибким для настройки и поддержки специфики деятельности различных команд разработчиков.

Создание UML началось в октябре 1994 г., когда Джим Рамбо и ГрадиБуч из RationalSoftwareCorporation стали работать над объединением своих методов OMT и Booch. В настоящее время консорциум пользователей UML Partners включает в себя представителей таких грандов информационных технологий, как RationalSoftware, Microsoft, IBM, Hewlett-Packard, Oracle, DEC, Unisys, IntelliCorp, PlatinumTechnology.

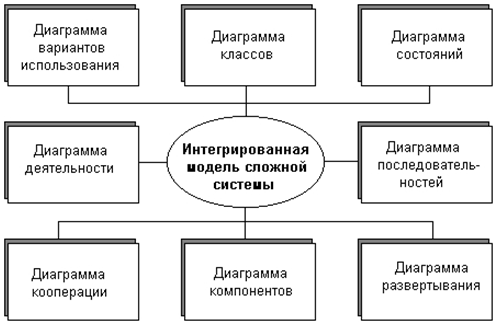
*UML* представляет собой *объектно-ориентированный* язык моделирования, обладающий следующими основными характеристиками:

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

UML — это стандартная нотация визуального моделирования программных систем, принятая консорциумом ObjectManagingGroup (OMG) осенью 1997 г., и на сегодняшний день она поддерживается многими объектно-ориентированными CASE-продуктами.

UML включает внутренний набор средств моделирования, которые сейчас приняты во многих методах и средствах моделирования. Эти концепции необходимы в большинстве прикладных задач, хотя не каждая концепция необходима в каждой части каждого приложения. Пользователям языка предоставлены возможности:

* строить модели на основе средств ядра, без использования механизмов расширения для большинства типовых приложений;
* добавлять при необходимости новые элементы и условные обозначения, если они не входят в ядро, или специализировать компоненты, систему условных обозначений (нотацию) и ограничения для конкретных предметных областей.



#### Рисунок 1 - Интегрированная модель сложной системы в нотации языка UML

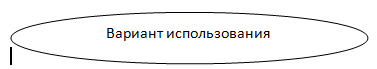
Стандарт UML предлагает  следующий  набор  диаграмм для моделирования:

* диаграммы  вариантов  использования (usecasediagrams) – для моделирования  бизнес-процессов  организации  и  требований к создаваемой системе);
* диаграммы  классов (classdiagrams) –  для  моделирования статической структуры  классов системы и связей между ними;
* диаграммы поведения системы (behaviordiagrams):
  + диаграммы взаимодействия (interaction diagrams):
    - диаграммы последовательности (sequence diagrams) и
    - кооперативные  диаграммы (collaborationdiagrams) – для моделирования  процесса  обмена  сообщениями между объектами;
  + диаграммы  состояний (statechartdiagrams)  – для моделирования  поведения объектов  системы  при  переходе из одного состояния в другое;
  + диаграммы  деятельностей (activitydiagrams) – для моделирования  поведения  системы  в  рамках  различных вариантов использования, или моделирования деятельностей;
* диаграммы реализации (implementationdiagrams):
  + диаграммы  компонентов (componentdiagrams) – для моделирования иерархии компонентов (подсистем) системы;
  + диаграммы  развертывания (deploymentdiagrams) – для моделирования физической архитектуры системы.

**Диаграммы вариантов использования**

Понятие  варианта  использования (usecase)  впервые  ввел Ивар Якобсон  и придал  ему  такую  значимость,  что  в  настоящее  время вариант  использования  превратился  в  основной  элемент  разработки и планирования проекта.

Вариант  использования  представляет  собой  последовательность действий (транзакций),  выполняемых  системой  в  ответ  на  событие, инициируемое  некоторым  внешним  объектом (действующим  лицом). Вариант  использования  описывает  типичное  взаимодействие между пользователем  и  системой.  В  простейшем  случае  вариант использования определяется в процессе обсуждения с пользователем тех функций, которые он хотел бы реализовать. На языке UML вариант использования изображают следующим образом:



### Рисунок 2 - Вариант использования

Действующее лицо (actor) – это роль, которую пользователь играет по отношению к системе. Действующие лица представляют собой роли, а не  конкретных  людей  или  наименования  работ.  Несмотря  на  то,  что на диаграммах  вариантов  использования  они  изображаются  в  виде стилизованных  человеческих  фигурок,  действующее  лицо  может  также быть  внешней  системой,  которой  необходима  некоторая  информация от данной системы. Показывать на диаграмме действующих лиц следует только  в  том  случае,  когда  им  действительно  необходимы  некоторые варианты использования.  На языке UML действующие лица представляют в виде фигур:

Описание: http://unesco.kemsu.ru/study_work/method/po/UMK/lab_pract/lab04.13.gif

Рисунок 3 - Действующее лицо (актер)

Действующие  лица  делятся  на  три  основных  типа:

* пользователи;
* системы;
* другие системы, взаимодействующие с данной;
* время.

Время становится действующим лицом, если от него зависит запуск каких-либо событий в системе.

**Связи  между  вариантами  использования  и  действующими лицами**

В  языке UML  на  диаграммах  вариантов  использования поддерживается несколько  типов  связей  между  элементами  диаграммы. Это  связи  коммуникации (communication),  включения (include), расширения (extend) и обобщения (generalization).

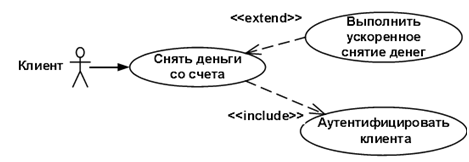
Связь  коммуникации –  это  связь  между  вариантом  использования и действующим лицом. На языке UML связи коммуникации показывают с помощью однонаправленной ассоциации (сплошной линии).



### Рисунок 4 - Пример связи коммуникации

Связь  включения  применяется  в  тех  ситуациях,  когда  имеется какой-либо фрагмент поведения системы, который повторяется более чем в одном  варианте  использования.  С  помощью  таких  связей  обычно моделируют  многократно  используемую  функциональность.

Связь  расширения  применяется  при  описании  изменений в нормальном поведении системы. Она позволяет варианту использования только  при необходимости  использовать  функциональные  возможности другого.



### Рисунок 5 - Пример связи включения и расширения

С  помощью  связи  обобщения  показывают,  что  у  нескольких действующих лиц имеются общие черты.

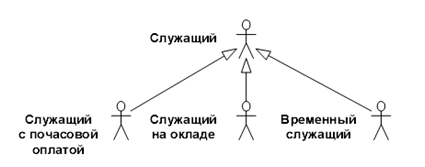


Рисунок 6 - Пример связи обобщения

**Диаграммы  взаимодействия (interactiondiagrams)**

Диаграммы взаимодействия (interactiondiagrams) 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.

*Сообщение (message)*  –  это  средство,  с  помощью  которого объект-отправитель запрашивает у объекта получателя выполнение одной из его операций.

*Информационное (informative)  сообщение* –  это  сообщение, снабжающее объект-получатель некоторой информацией для обновления его состояния.

*Сообщение-запрос (interrogative)* – это сообщение, запрашивающее выдачу некоторой информации об объекте-получателе.

*Императивное (imperative)  сообщение* –  это  сообщение, запрашивающее у объекта-получателя выполнение некоторых действий.

Существует  два  вида  диаграмм  взаимодействия:  диаграммы последовательности (sequencediagrams)  и  кооперативные  диаграммы (collaborationdiagrams).

**Диаграмма последовательности (sequencediagrams)**

Диаграмма последовательности отражает поток событий, происходящих в рамках варианта использования.

Все действующие лица показаны  в верхней  части  диаграммы. Стрелки  соответствуют сообщениям, передаваемым между действующим лицом и объектом или между объектами для выполнения требуемых функций.

На  диаграмме  последовательности  объект  изображается  в  виде прямоугольника,  от  которого  вниз  проведена  пунктирная  вертикальная линия.  Эта  линия  называется  линией  жизни (lifeline)  объекта.  Она представляет  собой  фрагмент  жизненного  цикла  объекта  в  процессе взаимодействия.

Каждое  сообщение  представляется  в  виде  стрелки  между  линиями жизни  двух  объектов. Сообщения  появляются  в  том  порядке,  как  они показаны  на странице  сверху  вниз.  Каждое  сообщение  помечается  как минимум  именем  сообщения. При  желании  можно  добавить  также аргументы и некоторую управляющую информацию. Можно показать самоделегирование (self-delegation) – сообщение, которое объект посылает самому себе, при этом стрелка сообщения указывает на ту же самую линию жизни.

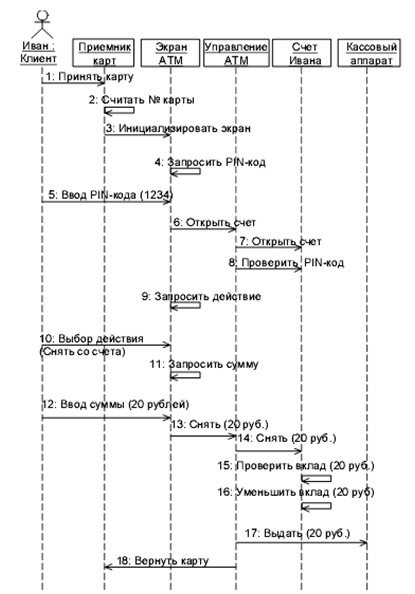


Рисунок 7 - Пример диаграммы последовательности

**Диаграмма кооперации (collaborationdiagram)**

Диаграммы кооперации  отображают поток  событий  через  конкретный  сценарий варианта  использования, упорядочены по времени,  а  кооперативные  диаграммы  больше  внимания  заостряют на связях  между  объектами.

На диаграмме кооперации представлена вся та информация, которая есть и на диаграмме последовательности, но кооперативная диаграмма по-другому  описывает  поток  событий. Из  нее  легче  понять  связи между объектами, однако, труднее уяснить последовательность событий.

На кооперативной  диаграмме  так  же,  как  и  на  диаграмме последовательности,  стрелки  обозначают  сообщения,  обмен  которыми осуществляется в рамках данного варианта использования. Их временная последовательность указывается путем нумерации сообщений.

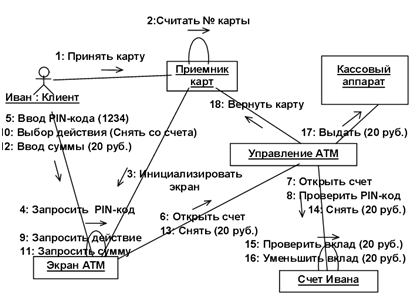


Рисунок 8 - Пример диаграммы кооперации

**Диаграммы классов**

**Общие сведения**

Диаграмма классов 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 UML - это граф, узлами которого являются элементы статической структуры проекта (классы, интерфейсы), а дугами - отношения между узлами (ассоциации, наследование, зависимости).

На диаграмме классов изображаются следующие элементы:

* Пакет (package) - набор элементов модели, логически связанных между собой;
* Класс (class) - описание общих свойств группы сходных объектов;
* Интерфейс (interface) - абстрактный класс, задающий набор операций, которые объект произвольного класса, связанного с данным интерфейсом, предоставляет другим объектам.

**Класс**

Класс - это группа сущностей (объектов), обладающих сходными свойствами, а именно, данными и поведением. Отдельный представитель некоторого класса называется объектом класса или просто объектом.

Под поведением объекта в UML понимаются любые правила взаимодействия объекта с внешним миром и с данными самого объекта.

На диаграммах класс изображается в виде прямоугольника со сплошной границей, разделенного горизонтальными линиями на 3 секции:

* Верхняя секция (секция имени) содержит имя класса и другие общие свойства (в частности, стереотип).
* В средней секции содержится список атрибутов
* В нижней - список операций класса, отражающих его поведение (действия, выполняемые классом).

Любая из секций атрибутов и операций может не изображаться (а также обе сразу). Для отсутствующей секции не нужно рисовать разделительную линию и как-либо указывать на наличие или отсутствие элементов в ней.

На усмотрение конкретной реализации могут быть введены дополнительные секции, например, исключения (Exceptions).

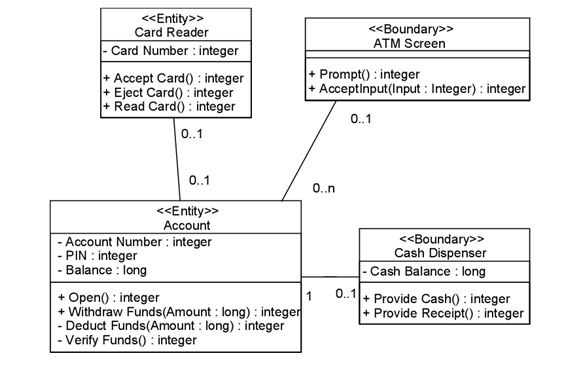


Рисунок 9 - Пример диаграммы классов

Стереотипы классов

Стереотипы классов –  это  механизм,  позволяющий  разделять  классы на категории.

В языке UML определены три основных стереотипа классов:

* Boundary (граница);
* Entity (сущность);
* Control (управление).

Граничные классы

Граничными  классами (boundaryclasses)  называются  такие  классы, которые  расположены  на  границе  системы  и  всей  окружающей  среды. Это экранные  формы,  отчеты,  интерфейсы  с  аппаратурой (такой  как принтеры или сканеры) и интерфейсы с другими системами.

Чтобы  найти  граничные  классы,  надо  исследовать  диаграммы вариантов использования. Каждому взаимодействию между действующим лицом и вариантом использования должен соответствовать, по  крайней мере, один граничный класс. Именно такой класс позволяет действующему лицу взаимодействовать с системой.

Классы-сущности

Классы-сущности (entityclasses)  содержат  хранимую  информацию. Они  имеют  наибольшее  значение  для  пользователя,  и  потому в их названиях часто используют термины из предметной области. Обычно для каждого класса-сущности создают таблицу в базе данных.

Управляющие классы

Управляющие  классы (controlclasses)  отвечают  за  координацию действий  других  классов.  Обычно  у  каждого  варианта  использования имеется один  управляющий класс, контролирующий последовательность событий  этого  варианта  использования.  Управляющий  класс  отвечает за координацию, но сам не несет в себе никакой функциональности, так как остальные классы не посылают ему большого количества сообщений. Вместо этого он сам посылает множество сообщений. Управляющий класс просто делегирует ответственность другим классам, по этой причине его часто называют классом-менеджером.

В  системе  могут  быть  и  другие  управляющие  классы,  общие для нескольких  вариантов  использования.  Например,  может  быть  класс SecurityManager(менеджер  безопасности),  отвечающий  за  контроль событий, связанных с безопасностью. Класс TransactionManager (менеджер транзакций)  занимается  координацией  сообщений,  относящихся к транзакциям  с  базой  данных.  Могут  быть  и  другие  менеджеры для работы с другими элементами функционирования системы, такими как разделение  ресурсов,  распределенная  обработка  данных  или  обработка ошибок.

Помимо  упомянутых  выше  стереотипов  можно  создавать  и свои собственные.

Атрибуты

Атрибут – это элемент информации, связанный с классом. Атрибуты хранят инкапсулированные данные класса.

Так как атрибуты содержатся внутри класса, они скрыты от других классов. В связи с этим может понадобиться указать, какие классы имеют право читать и изменять атрибуты. Это свойство называется видимостью атрибута (attributevisibility).

У  атрибута  можно  определить  четыре  возможных  значения  этого параметра:

* Public (общий, открытый). Это значение видимости предполагает, что атрибут будет виден всеми остальными классами. Любой класс может просмотреть или изменить значение атрибута.  В соответствии  с  нотацией UML  общему  атрибуту предшествует знак « + ».
* Private (закрытый,  секретный).  Соответствующий  атрибут не виден  никаким  другим  классом.  Закрытый  атрибут  обозначается  знаком  « – » в соответствии с нотацией UML.
* Protected (защищенный). Такой атрибут доступен только самому классу и его потомкам. Нотация UML для защищенного атрибута – это знак « # ».
* PackageorImplementation (пакетный). Предполагает, что данный атрибут является общим, но только в пределах его пакета. Этот  тип  видимости  не  обозначается никаким специальным значком.

В  общем  случае,  атрибуты  рекомендуется  делать  закрытыми  или защищенными. Это позволяет лучше контролировать сам атрибут и код.

С помощью  закрытости  или  защищенности  удается  избежать  ситуации, когда  значение  атрибута  изменяется  всеми  классами  системы.  Вместо этого  логика  изменения  атрибута  будет  заключена  в  том  же  классе, что и сам  этот  атрибут.  Задаваемые  параметры  видимости  повлияют на генерируемый код.

**Операции**

Операции  реализуют  связанное  с  классом  поведение.  Операция включает  три  части –  имя,  параметры  и  тип  возвращаемого  значения.

Параметры –  это  аргументы,  получаемые  операцией «на  входе».  Тип возвращаемого значения относится к результату действия операции.

На диаграмме классов можно показывать как имена операций, так и имена  операций  вместе  с  их  параметрами  и  типом  возвращаемого значения.  Чтобы  уменьшить  загруженность  диаграммы,  полезно бывает на некоторых из них показывать только имена операций, а на других их полную сигнатуру.

В языке UML операции имеют следующую нотацию:

*Имя Операции (аргумент: тип данных аргумента, аргумент2:тип данных аргумента2,...): тип возвращаемого значения*

Следует рассмотреть четыре различных типа операций:

* Операции реализации;
* Операции управления;
* Операции доступа;
* Вспомогательные операции.

**Операции реализации**

Операции реализации (implementoroperations) реализуют некоторые бизнес-функции.  Такие  операции  можно  найти,  исследуя  диаграммы взаимодействия.  Диаграммы  этого  типа  фокусируются  на  бизнес-функциях, и каждое сообщение диаграммы, скорее всего, можно соотнести с операцией реализации.

Каждая  операция  реализации  должна  быть  легко  прослеживаема до соответствующего  требования.  Это  достигается  на  различных  этапах моделирования.  Операция  выводится  из  сообщения  на  диаграмме взаимодействия,  сообщения  исходят  из  подробного  описания  потока событий,  который  создается  на  основе  варианта  использования, а последний – на основе требований. Возможность проследить всю эту цепочку  позволяет  гарантировать,  что  каждое  требование  будет реализовано  в  коде,  а  каждый  фрагмент  кода  реализует  какое-то требование.

**Операции управления**

Операции  управления (manageroperations)  управляют  созданием  и уничтожением  объектов.  В  эту  категорию  попадают  конструкторы  и деструкторы классов.

**Операции доступа**

Атрибуты  обычно  бывают  закрытыми  или  защищенными.  Тем не менее, другие классы иногда должны просматривать или изменять их значения. Для этого существуют операции доступа (accessoperations). Такой подход дает возможность безопасно инкапсулировать атрибуты внутри  класса,  защитив  их  от  других  классов,  но  все  же  позволяет осуществить к ним контролируемый доступ. Создание операций Get и Set (получения и изменения значения) для каждого атрибута класса является стандартом.

**Вспомогательные операции**

Вспомогательными (helperoperations)  называются  такие  операции класса, которые необходимы ему для выполнения его ответственностей, но о  которых  другие  классы  не  должны  ничего  знать.  Это  закрытые и защищенные операции класса.

Чтобы идентифицировать операции, выполните следующие действия:

1. Изучите  диаграммы  последовательности  и  кооперативные диаграммы.  Большая  часть  сообщений  на  этих  диаграммах является операциями реализации. Рефлексивные сообщения будут вспомогательными операциями.
2. Рассмотрите  управляющие  операции.  Может  потребоваться добавить конструкторы и деструкторы.
3. Рассмотрите  операции  доступа.  Для  каждого  атрибута  класса, с которым  должны  будут  работать  другие  классы,  надо  создать операции Get и Set.

**Связи**

Связь  представляет  собой  семантическую  взаимосвязь  между классами. Она дает классу возможность узнавать об атрибутах, операциях и связях другого класса. Иными словами, чтобы один класс мог послать сообщение другому на диаграмме последовательности или кооперативной диаграмме, между ними должна существовать связь.

Существуют четыре типа связей, которые могут быть установлены между классами: ассоциации, зависимости, агрегации и обобщения.

**Ассоциации**

Ассоциация (association) – это семантическая связь между классами. Их рисуют на диаграмме классов в виде обыкновенной линии.



Рисунок 10 - Связь ассоциация

Ассоциации  могут  быть  двунаправленными,  как  в  примере,  или однонаправленными. На языке UML двунаправленные ассоциации рисуют в виде простой линии без стрелок или со стрелками с обеих ее сторон. На однонаправленной  ассоциации  изображают  только  одну  стрелку, показывающую ее направление.

Направление  ассоциации  можно  определить,  изучая  диаграммы последовательности  и  кооперативные  диаграммы.  Если  все  сообщения на них отправляются только одним классом и принимаются только другим классом,  но  не  наоборот,  между  этими  классами  имеет  место однонаправленная  связь.  Если  хотя  бы  одно  сообщение  отправляется в обратную сторону, ассоциация должна быть двунаправленной.

Ассоциации  могут  быть  рефлексивными.  Рефлексивная  ассоциация предполагает,  что  один  экземпляр  класса  взаимодействует  с  другими экземплярами этого же класса.

**Зависимости**

Связи  зависимости (dependency)  также  отражают  связь между классами, но они всегда однонаправлены и показывают, что один класс  зависит  от определений,  сделанных  в  другом.  Например, класс A использует методы класса B. Тогда при изменении класса B необходимо произвести соответствующие изменения в классе A.

Зависимость изображается пунктирной линией, проведенной между двумя элементами диаграммы, и считается, что элемент, привязанный к концу стрелки, зависит от элемента, привязанного к началу этой стрелки.

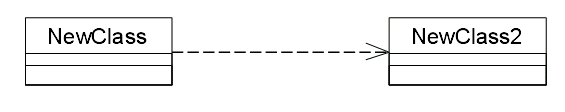


Рисунок 11 - Связь зависимость

При генерации кода для этих классов к ним не будут добавляться новые  атрибуты.  Однако, будут  созданы  специфические  для  языка операторы, необходимые для поддержки связи.

**Агрегации**

Агрегации (aggregations)  представляют  собой  более  тесную  форму ассоциации. Агрегация – это связь между целым и его частью. Например, у вас может быть класс Автомобиль, а также классы Двигатель, Покрышки и  классы  для  других  частей  автомобиля.  В  результате  объект  класса Автомобиль будет состоять из объекта класса Двигатель, четырех объектов Покрышек  и  т. д.  Агрегации  визуализируют  в  виде  линии  с  ромбиком у класса, являющегося целым:



Рисунок 12- Связь агрегация

В  дополнение  к  простой  агрегации UML  вводит  более  сильную разновидность  агрегации,  называемую  композицией.  Согласно композиции,  объект-часть  может  принадлежать  только  единственному целому,  и,  кроме того,  как  правило,  жизненный  цикл  частей  совпадает с циклом целого: они живут и умирают вместе с ним. Любое удаление целого распространяется на его части.

Такое  каскадное  удаление  нередко  рассматривается  как  часть определения агрегации, однако оно всегда подразумевается в том случае, когда множественность роли составляет  1..1; например, если необходимо удалить Клиента, то это удаление должно распространиться и на Заказы (и, в свою очередь, на Строки заказа).

**Обобщения (Наследование)**

Обобщение (наследование) - это отношение типа общее-частное между элементами модели. С  помощью  обобщений (generalization)  показывают  связи наследования  между  двумя  классами.  Большинство  объектно-ориентированных  языков  непосредственно  поддерживают  концепцию наследования.  Она  позволяет  одному  классу  наследовать  все  атрибуты, операции и связи другого. Наследование пакетов означает, что в пакете-наследнике все сущности пакета-предка будут видны под своими собственными именами (т.е. пространства имен объединяются). Наследование показывается сплошной линией, идущей от класса-потомка к классу-предку (в терминологии ООП - от потомка к предку, от сына к отцу, или от подкласса к суперклассу).  Со стороны более общего элемента рисуется большой полый треугольник.

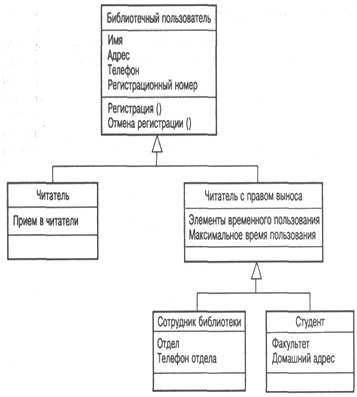


Рисунок 13 - Пример связи наследование

Помимо  наследуемых,  каждый  подкласс  имеет  свои  собственные уникальные атрибуты, операции и связи.

**Множественность**

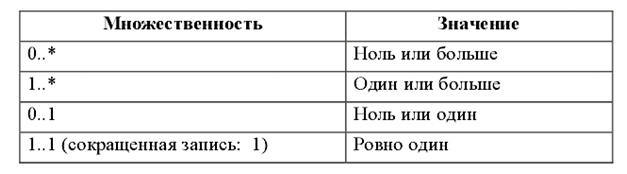
Множественность (multiplicity)  показывает,  сколько  экземпляров одного  класса  взаимодействуют  с  помощью  этой  связи  с  одним экземпляром другого класса в данный момент времени.

Например,  при  разработке  системы  регистрации  курсов в университете  можно  определить  классы Course (курс)  и Student (студент). Между ними установлена связь: у курсов могут быть студенты, а у студентов – курсы. Вопросы, на который должен ответить параметр множественности: «Сколько  курсов  студент  может  посещать  в  данный момент? Сколько студентов может за раз посещать один курс?»

Так  как  множественность  дает  ответ  на  оба  эти  вопроса,  её индикаторы устанавливаются на обоих концах линии связи. В примере регистрации  курсов  мы  решили,  что  один  студент  может  посещать от нуля до четырех  курсов,  а  один  курс  могут  слушать  от  0  до 20 студентов.

В  языке UML  приняты  определенные нотации  для  обозначения множественности.

Таблица 1 - Обозначения множественности связей в UML



**Имена связей**

Связи можно уточнить с помощью имен связей или ролевых имен. Имя связи – это обычно глагол или глагольная фраза, описывающая, зачем она нужна. Например, между классом Person (человек) и классом Company (компания) может существовать ассоциация. Можно задать в связи с этим вопрос,  является  ли  объект  класса Person  клиентом  компании, её сотрудником  или  владельцем?  Чтобы  определить  это,  ассоциацию можно назвать «employs» (нанимает):



Рисунок 14 - Пример имен связей

**Роли**

Ролевые имена применяют в связях ассоциации или агрегации вместо имен для описания того, зачем эти связи нужны. Возвращаясь к примеру с классами Person и Company, можно сказать, что класс Person играет роль сотрудника  класса Company.  Ролевые  имена –  это  обычно  имена существительные  или  основанные  на  них  фразы,  их  показывают на диаграмме  рядом  с  классом,  играющим  соответствующую  роль.  Как правило,  пользуются  или  ролевым  именем,  или  именем  связи, но не обоими сразу. Как и имена связей, ролевые имена не обязательны, их дают, только если цель связи не очевидна. Пример ролей приводится ниже:

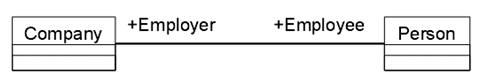


Рисунок 15 - Пример ролей  связей

**Пакет. Механизм пакетов**

В контексте диаграмм классов, пакет - это вместилище для некоторого набора классов и других пакетов. Пакет является самостоятельным пространством имен.

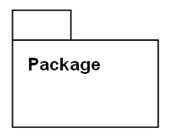


Рисунок 16 - Обозначение пакета в UML

В UML нет каких-либо ограничений на правила, по которым разработчики могут или должны группировать классы в пакеты. Но есть некоторые стандартные случаи, когда такая группировка уместна, например, тесно взаимодействующие классы, или более общий случай - разбиение системы на подсистемы.

Пакет физически содержит сущности, определенные в нем (говорят, что "сущности принадлежат пакету"). Это означает, что если будет уничтожен пакет, то будут уничтожено и все его содержимое.

**Существует несколько наиболее распространенных подходов  к  группировке.**

Во-первых,  можно  группировать  их по стереотипу.  В  таком  случае  получается  один  пакет  с  классами-сущностями,  один  с  граничными  классами,  один  с  управляющими классами  и  т.д.  Этот  подход  может  быть  полезен  с  точки  зрения размещения готовой системы, поскольку все находящиеся на клиентских машинах пограничные классы уже оказываются в одном пакете.

Другой  подход  заключается  в  объединении  классов по их функциональности.  Например,  в  пакете Security (безопасность) содержатся все классы, отвечающие за безопасность приложения. В таком случае  другие  пакеты  могут  называться EmployeeMaintenance (Работа с сотрудниками), Reporting (Подготовка  отчетов)  и ErrorHandling (Обработка  ошибок).  Преимущество  этого  подхода  заключается в возможности повторного использования.

Механизм пакетов применим к любым элементам модели, а не только к классам.  Если  для  группировки  классов  не  использовать  некоторые эвристики,  то  она  становится  произвольной.  Одна  из  них,  которая в основном используется в UML, – это зависимость. Зависимость между двумя  пакетами  существует  в  том  случае,  если  между  любыми  двумя классами  в пакетах  существует  любая  зависимость.

Таким  образом, диаграмма  пакетов представляет  собой  диаграмму, содержащую пакеты классов и зависимости между ними. Строго говоря, пакеты и зависимости являются элементами диаграммы классов, то есть диаграмма пакетов – это форма диаграммы классов.

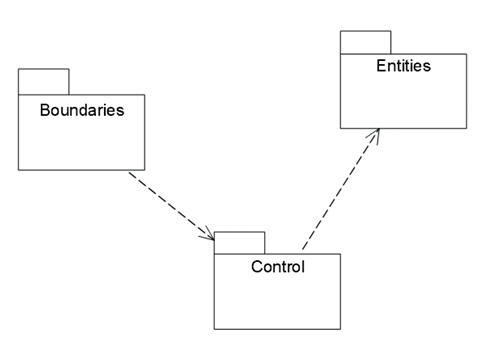


Рисунок 17 - Пример диаграммы пакетов

Зависимость между двумя элементами имеет место в том случае, если изменения  в  определении  одного  элемента  могут  повлечь  за  собой изменения в другом. Что касается классов, то причины для зависимостей могут быть самыми разными:

* один класс посылает сообщение другому;
* один класс включает часть данных другого класса; один класс использует другой в качестве параметра операции.

Если класс меняет свой интерфейс, то любое сообщение, которое он посылает, может утратить свою силу.

Пакеты не дают ответа на вопрос, каким образом можно уменьшить количество зависимостей в вашей системе, однако они помогают выделить эти зависимости, а после того, как они все окажутся на виду, остается только  поработать  над  снижением  их  количества.  Диаграммы  пакетов можно  считать  основным  средством  управления  общей  структурой системы.

Пакеты  являются  жизненно  необходимым  средством  для  больших проектов.  Их  следует  использовать  в  тех  случаях,  когда  диаграмма классов,  охватывающая  всю  систему  в  целом  и  размещенная на единственном листе бумаги формата А4, становится нечитаемой.

**Диаграммы состояний**

Диаграммы  состояний  определяют  все  возможные  состояния, в которых может находиться конкретный объект, а также процесс смены состояний  объекта  в  результате  наступления  некоторых  событий.

Существует  много  форм  диаграмм  состояний,  незначительно отличающихся  друг  от  друга  семантикой.

На диаграмме имеются два специальных состояния – начальное (start) и  конечное (stop).  Начальное  состояние  выделено  черной  точкой,  оно соответствует  состоянию  объекта,  когда  он  только  что  был  создан. Конечное  состояние  обозначается  черной  точкой  в  белом  кружке,  оно соответствует  состоянию  объекта  непосредственно  перед  его уничтожением. На диаграмме состояний может быть одно и только одно начальное  состояние.  В  то  же  время,  может  быть  столько  конечных состояний,  сколько  вам  нужно,  или  их  может  не  быть  вообще.  Когда объект находится в каком-то конкретном состоянии, могут выполняться различные процессы. Процессы, происходящие, когда объект  находится  в  определенном  состоянии,  называются  действиями (actions).

С  состоянием  можно  связывать  данные  пяти  типов:  деятельность, входное  действие,  выходное  действие,  событие  и  история  состояния.

**Деятельность**

Деятельностью (activity)  называется  поведение,  реализуемое объектом, пока он находится в данном состоянии.  Деятельность –  это  прерываемое  поведение.  Оно  может выполняться  до  своего  завершения,  пока  объект  находится  в  данном состоянии,  или  может  быть  прервано  переходом  объекта  в  другое состояние. Деятельность изображают внутри самого состояния, ей должно предшествовать слово do (делать) и двоеточие.

**Входное действие**

Входным  действием (entryaction)  называется  поведение,  которое выполняется, когда объект переходит в данное состояние. Данное действие осуществляется не после того, как объект перешел в это состояние, а, скорее, как часть этого  перехода.  В  отличие  от  деятельности,  входное  действие рассматривается как непрерываемое. Входное  действие  также  показывают  внутри  состояния,  ему предшествует слово entry (вход) и двоеточие.

**Выходное действие**

Выходное  действие (exitaction)  подобно  входному.  Однако,  оно осуществляется  как  составная  часть  процесса  выхода  из  данного состояния. Оно  является  частью  процесса  такого перехода. Как и входное, выходное действие является непрерываемым.

Выходное действие изображают внутри состояния, ему предшествует слово exit (выход) и двоеточие.

Поведение объекта во время деятельности, при входных и выходных действиях может включать отправку события другому объекту. В  этом  случае  описанию  деятельности, входного  действия  или  выходного  действия  предшествует  знак  « ^ ».

Соответствующая строка на диаграмме выглядит как

*Do: ^Цель.Событие (Аргументы)*

Здесь  Цель –  это  объект,  получающий  событие,  Событие –  это посылаемое сообщение, а Аргументы являются параметрами посылаемого сообщения.

Деятельность  может  также  выполняться  в  результате  получения объектом  некоторого  события.   При  получении  некоторого  события  выполняется определенная деятельность.

Переходом (Transition) называется перемещение из одного состояния в другое.  Совокупность  переходов  диаграммы  показывает,  как  объект может  перемещаться  между  своими  состояниями.  На  диаграмме  все переходы изображают в виде стрелки, начинающейся на первоначальном состоянии и заканчивающейся последующим.

Переходы могут быть рефлексивными. Объект может перейти в то же состояние, в котором он в настоящий момент находится. Рефлексивные переходы изображают в виде стрелки, начинающейся и завершающейся на одном и том же состоянии.

У  перехода  существует  несколько  спецификаций.  Они  включают события,  аргументы,  ограждающие  условия,  действия  и  посылаемые события.

**События**

Событие (event) – это то, что вызывает переход из одного состояния в другое. Событие размещают на диаграмме вдоль линии перехода.

На диаграмме для отображения события можно использовать как имя операции,  так  и  обычную  фразу.

Большинство переходов должны иметь события, так как именно они, прежде всего, заставляют переход осуществиться. Тем не менее, бывают и автоматические переходы, не имеющие событий. При этом объект сам перемещается из одного состояния в другое со скоростью, позволяющей осуществиться входным действиям, деятельности и выходным действиям.

**Ограждающие условия**

Ограждающие условия (guardconditions) определяют, когда переход может,  а  когда  не  может  осуществиться.  В противном случае переход не осуществится.

Ограждающие  условия  изображают  на  диаграмме  вдоль  линии перехода после имени события, заключая их в квадратные скобки.

Ограждающие  условия  задавать  необязательно.  Однако  если существует  несколько  автоматических  переходов  из  состояния, необходимо  определить  для них  взаимно  исключающие  ограждающие условия. Это поможет читателю диаграммы понять, какой путь перехода будет автоматически выбран.

**Действие**

Действием (action),  как  уже  говорилось,  является  непрерываемое поведение, осуществляющееся как часть перехода. Входные и выходные действия показывают внутри состояний, поскольку они определяют, что происходит, когда объект входит или выходит из него. Большую часть действий,  однако,  изображают  вдоль  линии  перехода,  так  как  они не должны осуществляться при входе или выходе из состояния.

Действие рисуют вдоль линии перехода после имени события, ему предшествует косая черта.

Событие  или  действие  могут  быть  поведением  внутри  объекта, а могут  представлять  собой  сообщение,  посылаемое  другому  объекту. Если событие  или  действие  посылается  другому  объекту,  перед  ним на диаграмме помещают знак « ^ ».

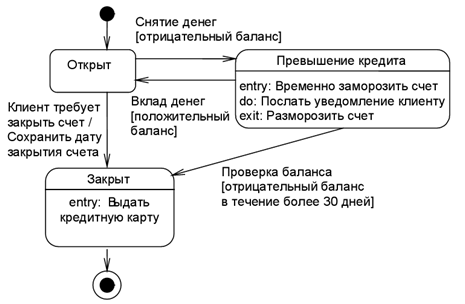


Рисунок 18 - Пример диаграммы состояний

Диаграммы  состояний  не  надо  создавать  для  каждого  класса,  они применяются  только  в  сложных  случаях.  Если  объект  класса  может существовать  в  нескольких  состояниях  и  в  каждом  из  них  ведет  себя по-разному, для него может потребоваться такая диаграмма.

1. Порядок выполнения работы
2. Изучить предлагаемый теоретический материал.
3. Постройте диаграмму вариантов использования для выбранной информационной системы **(Варианты заданий см. в Приложении 1).**
4. Выполните реализацию вариантов использования в терминах  взаимодействующих  объектов  и  представляющую  собой  набор диаграмм:

* диаграмм классов,  реализующих вариант использования;
* диаграмм  взаимодействия (диаграмм  последовательности  и кооперативных  диаграмм),  отражающих  взаимодействие  объектов  в процессе реализации варианта использования.

1. Разделить классы по пакетам, используя один из механизмов разбиения.
2. Постройте диаграмму состояний для конкретных объектов информационной системы.
3. Построить отчёт, включающий все полученные уровни модели, описание функциональных блоков, потоков данных, хранилищ и внешних объектов.

**В отчете следует указать:**

1. Цель работы
2. Введение
3. Программно-аппаратные средства, используемые при выполнении работы.
4. Основную часть (описание самой работы), выполненную согласно требованиям к результатам выполнения.
5. Заключение (выводы)

**Контрольные вопросы**

1. Дайте определение понятию «вариант использования».
2. Какие типы связи могут присутствовать на диаграмме вариантов использования?
3. Дайте определение понятию «действующее лицо».
4. Какие типы сообщений могут присутствовать на диаграммах взаимодействия?
5. Дайте определение понятию класс, объект класса.
6. Кем и для чего может быть использована диаграмма  размещения?

### Лабораторная работа №№10-11

### Тема: Работа с CASE – средствами проектирования программного обеспечения

**Цель работы:** изучение среды программного инструмента моделирования StarUML, поддерживающего UML, и приобретение навыков по проектированию системы.

**Теоретическая часть**

По мере завершения анализа фокус моделирования перемещается на проектирование. В принципе анализ и проектирование могут проводиться параллельно. Методология UP рекомендует не разделять функции аналитиков и проектировщиков. За разработку артефактов (таких как прецедент) – от требований через анализ и проектирование до реализации – должна отвечать одна команда

Проектная модель содержит все то же самое, что и аналитическая модель, но все артефакты в ней проработаны более основательно и должны включать детали реализации. Проектная модель базируется на аналитической модели и может считаться просто ее улучшенной и уточненной версией.

В общем случае проектные модели образуются:

- проектными подсистемами;

- проектными классами;

- интерфейсами;

- реализациями проектных прецедентов;

- диаграммой развертывания (в первом приближении).

Объектно-ориентированное проектирование, в частности методология UP, включает два вида деятельности: проектирование архитектуры системы и проектирование элементов системы.

Проектирование архитектуры системы ведется на всех этапах ЖЦ ПО, выполняется архитектором системы и включает в себя: 1) идентификацию архитектурных решений и механизмов, не­обходимых для проектирования системы; 2) анализ взаимодействий между классами анализа, выявление подсистем и интерфейсов; 3) формирование архитектурных уровней; 4) проектирование структуры потоков управления; 5) проектирование конфигурации системы.

Проектирование элементов системы выполняется проектировщиками и включает в себя: 1) уточнение описания вариантов использования; 2) проектирование классов; 3) проектирование баз данных (диаграммы развёртывания).

С другой стороны, технологический процесс проектирования предусматривает разработку или модификацию

- диаграмм состояний

- диаграмм деятельности.

*Архитектор* программного обеспечения в первую *очередь* обращает внимание на объекты *предметной области*. Программист же концентрируется на поведении этих объектов, пользуясь **классами**, к которым они принадлежат. Вот поэтому-то *диаграмма* классов и является одной из важнейших диаграмм *UML*. Она используется для документирования программных систем, и основным ее компонентом является **класс**.

Сокрытие от пользователя внутреннего устройства объектов называется *инкапсуляцией*. Если говорить более "научным" языком, то *инкапсуляция* - это защита отдельных элементов объекта, не затрагивающих существенных характеристик его как целого. *Инкапсуляция* нужна не только для того, чтобы создать иллюзию простоты объекта для пользователя (по словам Г. Буча).

В программировании *инкапсуляция* обеспечивается с помощью т. н. *модификаторов видимости*. С их помощью можно ограничить *доступ* к атрибутам и операциям объекта со стороны других объектов. Если *атрибут* или операция описаны с модификатором private, то *доступ* к ним можно получить только из *операции*, определенной в том же классе. Если же *атрибут* или операция описаны с модификатором видимости public, то к ним можно получить *доступ* из любой части программы.

Модификатор protected разрешает *доступ* только из операций этого же класса и классов, создаваемых на его основе. В языках программирования могут встречаться *модификаторы видимости*, ограничивающие *доступ* на более высоком уровне, например, к классам или их группам, однако смысл инкапсуляции от этого не изменяется. В *UML* атрибуты и *операции* с модификаторами доступа обозначаются специальными символами слева от их имен:

|  |  |
| --- | --- |
| **Символ** | **Значение** |
| + | public - открытый доступ |
| - | private - только из операций того же класса |

### Как использовать объекты класса?

Если уж говорить о защите объекта, то чтобы она действительно была эффективной, надо позаботиться о некоем стандартном и безопасном, не зависящим от языка программирования способе доступа к объекту. К тому же такой стандартный способ доступа должен быть простым и с точки зрения использования, и с точки зрения реализации.

Более строго, *интерфейс* - это логическая *группа* открытых ( public) операций объекта. Один и тот же *объект* может иметь несколько интерфейсов.

*Интерфейс* отражает внешние проявления объекта, показывает, каким образом осуществляется взаимодействие с ним, скрывая остальные детали, не имеющие отношения к процессу взаимодействия.

*Интерфейс* всегда реализуется некоторым классом, который в таком случае называют классом, *поддерживающим* *интерфейс*. Как мы уже говорили ранее, один и тот же *объект* может иметь несколько интерфейсов. Это означает, что *класс* этого объекта реализует все *операции* этих интерфейсов. Многие из существующих технологий программирования (например, *COM*, *CORBA*, *JavaBeans*) не только активно используют *механизм интерфейсов*, но и, по сути, полностью основаны на нем.

*Интерфейс* изображается на диаграммах несколькими способами. Первый и самый простой из них - это *класс* со стереотипом <<interface>> ([рис.](http://www.intuit.ru/studies/courses/1007/229/lecture/3176?page=1#image.3.3) 1):

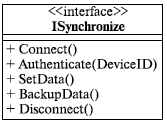


Рисунок 1 - Класс со стереотипом

Этот способ хорош, если нужно показать, какие именно *операции* предоставляет *интерфейс*. Если же такие подробности в данный момент не важны, предоставляемый *интерфейс* изображают в виде кружочка или, как говорят, "леденца" ( lollipop ) ([рис. 2](http://www.intuit.ru/studies/courses/1007/229/lecture/3176?page=1#image.3.4)):

http://www.intuit.ru/EDI/08_11_13_2/1383906499-13678/tutorial/356/objects/3/files/03_04.gif

Рисунок 2 - Класс со стереотипом

Обратите внимание на маленький значок на закладке папки ConduitSet. Это обозначение подсистемы, мы могли бы не рисовать его, а просто использовать стереотип <<subsystem>>. Впрочем, об этом мы еще поговорим.

И наконец, еще один способ изображения интерфейса. Он не является альтернативой описанным ранее способам, а используется для изображения интерфейсов, **требующихся** объекту для выполнения его работы. Обозначается он очень простым и логичным символом.([рис. 3](http://www.intuit.ru/studies/courses/1007/229/lecture/3176?page=1#image.3.5)):

http://www.intuit.ru/EDI/08_11_13_2/1383906499-13678/tutorial/356/objects/3/files/03_05.gif

Рисунок 3- Способ изображения интерфейса

Наблюдательный читатель уже, наверное, заметил, как логически совмещаются символы предоставляемого и требуемого интерфейсов.

Действительно, на диаграммах довольно часто можно увидеть такую картинку ([рис.4](http://www.intuit.ru/studies/courses/1007/229/lecture/3176?page=1#image.3.6)):

http://www.intuit.ru/EDI/08_11_13_2/1383906499-13678/tutorial/356/objects/3/files/03_06.gif

Рисунок 4 -Совмещение символов предоставляемого и требуемого интерфейсов

Да, кстати, вы заметили, что названия интерфейсов начинаются с буквы I? Эта традиция пошла из языка *Java*, и, как показывает практика, она весьма облегчает жизнь, если нужно, например, быстро разобраться в сложной диаграмме, составленной другим человеком.

### Всегда ли нужно создавать новые классы?

Начнем с вопроса, казалось бы, не имеющего никакого отношения к рассматриваемому вопросу, а именно - всегда ли нужно создавать новый *класс* для каждой новой задачи? *Правильный ответ*, конечно же, "нет". Это было бы странно и неэффективно. Могут быть ситуации, когда существующие классы по каким-либо причинам не устраивают архитектора, и тогда требуется создать новый *класс*. Следует, однако, избегать ситуаций, когда созданный *класс* (а точнее, его набор операций и атрибутов) практически повторяет существующий, лишь незначительно отличаясь от него. В общем случае, сказанное выше можно проиллюстрировать такой диаграммой ([рис. 5](http://www.intuit.ru/studies/courses/1007/229/lecture/3176?page=2#image.3.7)):

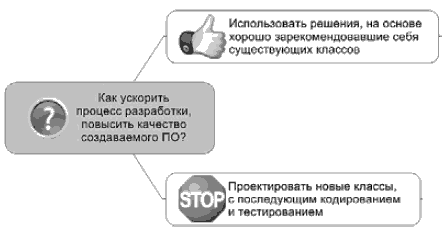


Рисунок 5 - Иллюстрация примера

В *дополнение* можно назвать несколько причин, почему стоит использовать уже существующие классы:

Во-первых, идя этим путем, мы пользуемся плодами ранее принятых решений. Действительно, если когда-то мы уже решили некоторую проблему, зачем начинать все "с нуля", повторяя уже однажды проделанные действия?

Во-вторых, таким образом, мы делаем решение мобильным и расширяемым. Используя уже существующие классы и создавая на их основе новые, мы можем развивать решение практически неограниченно, добавляя лишь необходимые нам в данный момент детали - атрибуты и *операции*.

В-третьих, существующие классы, как правило, хорошо отлажены и показали себя в работе. Разработчику не надо тратить время на *кодирование*, отладку, тестирование и т. д., - мы работаем с хорошо отлаженным и проверенным временем кодом, который зарекомендовал себя в других проектах и в котором уже выявлено и исправлено большинство ошибок.

Понятия **обобщения** или **генерализации**играют очень важную роль в *ООП*, являясь одним из его базовых принципов.

**Обобщение** – это *отношение* между более общей сущностью, называемой *суперклассом*, и ее конкретным воплощением, называемым *подклассом*. Иногда *обобщение* называют отношениями типа "является", имея в виду, что одни сущности (например, круг, квадрат, треугольник) являются воплощением более общей сущности (например, класса "*геометрическая фигура*"). При этом все атрибуты и *операции* суперкласса независимо от *модификаторов видимости* входят в состав подкласса.

*Обобщение* (или, как часто говорят, *наследование*) на диаграммах обозначается очень просто - незакрашенной треугольной стрелкой, направленной на *суперкласс* ([рис. 6](http://www.intuit.ru/studies/courses/1007/229/lecture/3176?page=2#image.3.8)).

Для того чтобы научиться эффективно моделировать *наследование*, обратимся к классикам, а именно к Г. Бучу. Он советует проводить эту процедуру в такой последовательности:

1. Найдите атрибуты, операции и обязанности, общие для двух или более классов из данной совокупности. Это позволит избежать ненужного дублирования структуры и функциональности объектов.
2. Вынесите эти элементы в некоторый общий суперкласс, а если такого не существует, то создайте новый класс.
3. Отметьте в модели, что подклассы наследуются от суперкласса, установив между ними отношение обобщения.

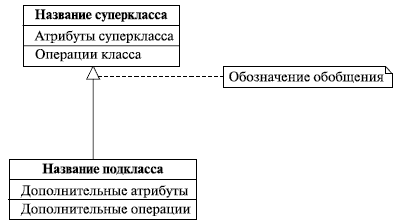


Рисунок 6 – Обозначение обобщения

А вот и пример применения этого подхода ([рис. 7](http://www.intuit.ru/studies/courses/1007/229/lecture/3176?page=2#image.3.9)):

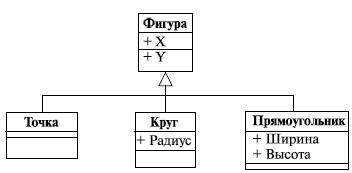


Рисунок 7 – Пример применения обобщения

Объекты разной природы (или говоря проще, разных классов) могут поддерживать один и тот же *интерфейс* именно так, как того ожидает *пользователь*. Примером тому может служить рассмотренная выше *диаграмма* с геометрическими фигурами. Все рассмотренные фигуры имеют, например, операцию рисования на экране. С точки зрения пользователя в каждом случае это одно и то же действие. Однако реализованы эти *операции* по-разному - ведь процедура изображения прямоугольника сильно отличается от подобной процедуры для круга. Но для пользователя это неважно: ведь *сигнатура*-то одна и та же. А возможно это благодаря еще одному из основных принципов *ООП* - **полиморфизму**. Как мы только что упомянули, работа механизма полиморфизма основана на совпадении сигнатуры метода, объявленного в интерфейсе, и сигнатуры самого метода. Методы внутри классов-потомков могут быть (и наверняка будут!) переопределены, их реализации будут различными, а сигнатуры останутся неизменными. Таким образом (и в этом легко ощутить мощь *ООП*), выполняя одни и те же *операции*, разные объекты могут вести себя по-разному.

*Полиморфизм* является основой для реализации *механизма интерфейсов* в языках программирования. Как только *пользователь* обращается к некоторой *операции* через *интерфейс*, определяется фактический *класс* объекта и вызывается соответствующая *операция класса*. Примеры полиморфизма можно увидеть в самых обыденных вещах, которыми мы пользуемся в повседневной жизни. Оглянитесь вокруг - мир построен по *ООП*,  Например, всем привычная кредитная карточка, является интерфейсом для доступа к банковскому счету через банкомат (и не только), одинаково работает в любой стране, вот только ведет себя чуть-чуть по-разному, т. к. банкомат выдает деньги в местной валюте. Пример не очень корректный, но зато очень наглядный.

*Инкапсуляция*, *наследование* и *полиморфизм*, с которыми мы только что познакомились, являются теми самыми тремя китами, на которых держится *ООП*. Если вы поняли суть этих базовых принципов и осознали их истинную мощь, вы прошли большую часть пути, ведущего к полному овладению *ООП* как наиболее адекватной методикой описания (так и тянет сказать "проектирования") окружающего нас мира.

### Отношения между классами

Ни один из объектов окружающего нас мира не существует сам по себе. Птицы летают потому, что есть воздух, на который опираются их крылья. Каждый из нас связан с массой других людей разнообразными родственными, профессиональными и другими связями, предполагающими различные типы отношений. Точно так же и классы связаны между собой. И чтобы в полной мере овладеть *ООП*, нам необходимо понять суть этих отношений и научиться их идентифицировать.

Мы сказали, что объекты находятся в определенных отношениях друг с другом. Один из типов таких отношений – это **зависимость**. Думаем, суть такого отношения понятна уже из его названия - зависимость возникает тогда, когда реализация класса одного объекта зависит от спецификации операций класса другого объекта. И если изменится спецификация операций этого класса, нам неминуемо придется вносить изменения и в зависимый *класс*. Приведем простой пример, опять-таки взятый из нашей повседневности. Иногда к нам в руки попадают видеофайлы, воспроизвести которые "с лету" не удается. Почему? Правильно, потому что на компьютере не установлены соответствующие кодеки. То есть операция "Воспроизведение", реализуемая программой-медиаплеером, зависит от *операции* "Декомпрессия", реализуемой кодеком. Если спецификация *операции*"Декомпрессия" изменится, придется менять код медиаплеера, иначе он просто не сможет работать с каким-то кодеком и, в лучшем случае, завершит свою работу с ошибкой. А вот так зависимость между классами изображается в *UML* ([рис. 8](http://www.intuit.ru/studies/courses/1007/229/lecture/3176?page=3#image.3.10)):

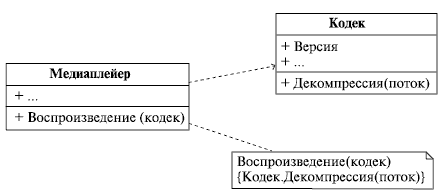


Рисунок 8 – Изображение зависимости между классами

Стоит отметить, что зависимости на диаграммах изображают далеко не всегда, а только в тех случаях, когда их *отображение* является важным для понимания модели. Часто зависимости лишь подразумеваются, т. к. логически следуют из природы классов.

Другой вид отношений между объектами - это **ассоциация**. Это просто *связь* между объектами, по которой можно между ними перемещаться. *Ассоциация* может иметь имя, показывающее природу отношений между объектами, при этом в имени может указываться *направление* чтения связи при помощи треугольного маркера. Однонаправленная *ассоциация* может изображаться стрелкой. Проиллюстрируем сказанное примерами ([рис. 9](http://www.intuit.ru/studies/courses/1007/229/lecture/3176?page=3#image.3.11)):

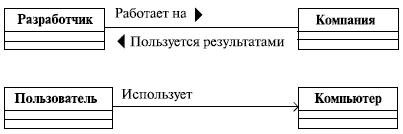


Рисунок 9 – Пример изображения однонаправленной ассоциации

Кроме направления ассоциации, мы можем указать на диаграмме *роли*, которые каждый *класс* играет в данном отношении, и*кратность*, то есть количество объектов, связанных отношением ([рис. 10](http://www.intuit.ru/studies/courses/1007/229/lecture/3176?page=3#image.3.12)):

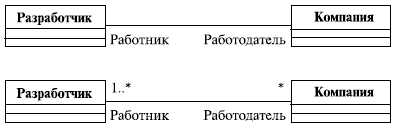


Рисунок 10 – Пример изображения кратности

И насчет ролей, и насчет кратности на этой диаграмме все понятно - человек может вообще не работать, работать в одной или более компаниях, а вот компании в любом случае нужен хотя бы один сотрудник. Кстати, о кратности. *Ассоциация* может объединять три и более класса. В этом случае она называется **n-арной** и изображается ромбом на пересечении линий, как показано на этой диаграмме, позаимствованной нами из ZicomMentor([рис. 11](http://www.intuit.ru/studies/courses/1007/229/lecture/3176?page=3#image.3.13)):

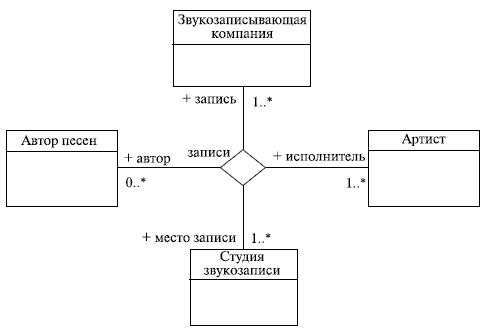


Рисунок 11 – Пример изображения n-арной  ассоциации

Ранее мы говорили, что *ассоциация* - это "просто *связь*" между объектами. На самом деле, в реальности связи бывают "просто связями" крайне редко. Обычно при ближайшем рассмотрении под ассоциацией понимается более сложное *отношение* между классами, например, *связь* типа "часть-целое". Такой вид ассоциации называется **ассоциацией с агрегированием**. В этом случае один *класс* имеет более высокий статус (целое) и состоит из низших по статусу классов (частей). При этом выделяют простое и композитное *агрегирование* и говорят о собственно **агрегации** и **композиции**. Простая *агрегация* предполагает, что части, отделенные от целого, могут продолжать свое существование независимо от него. Под композитным же агрегированием понимается ситуация, когда целое владеет своими частями и их время жизни соответствует времени жизни целого, т. е. независимо от целого части существовать не могут. Примеры этих видов ассоциаций и их обозначений в *UML* можно увидеть на следующей диаграмме ([рис. 12](http://www.intuit.ru/studies/courses/1007/229/lecture/3176?page=3#image.3.14)).

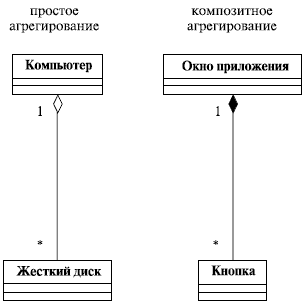


Рисунок 12 – Пример изображения агрегации и композиции

*Винчестер* можно вынуть из компьютера и установить в новый *компьютер*или в USB-карман, т. е. существование жесткого диска с разборкой системного блока не заканчивается. А вот кнопки без окон обычно существовать не могут - с закрытием окна кнопки также исчезают.

И, наконец, еще одна важная вещь, касающаяся ассоциации. В отношении между двумя классами сама *ассоциация* тоже может иметь свойства и, следовательно, тоже может быть представлена в виде класса. Пример ([рис. 13](http://www.intuit.ru/studies/courses/1007/229/lecture/3176?page=3#image.3.15)):

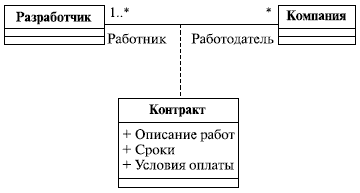


Рисунок 13 – Пример представления ассоциации в виде класса

Действительно, перед началом трудовых отношений работник и работодатель подписывают между собой контракт, который имеет такие атрибуты, как, например, описание *работ*, сроки их выполнения, порядок оплаты и т. д.

А вот более сложный, но, опять-таки, взятый из реальной жизни пример моделирования отношений между классами, позаимствованный нами из ZicomMentor([рис. 14](http://www.intuit.ru/studies/courses/1007/229/lecture/3176?page=3#image.3.16)):

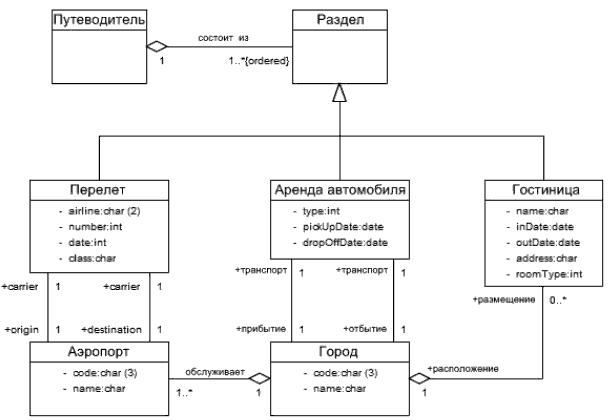


Рисунок 14 – Пример моделирования отношений между классами

И наконец, *доказательство* того, что *UML* можно использовать для чего угодно, в том числе и для записи сказок: *диаграмма*, описывающая предметную область сказки о Курочке Рябе и взятая с сайта конкурса шуток на *UML* (<http://www.umljokes.com/>) ([рис. 15](http://www.intuit.ru/studies/courses/1007/229/lecture/3176?page=3#image.3.17)):

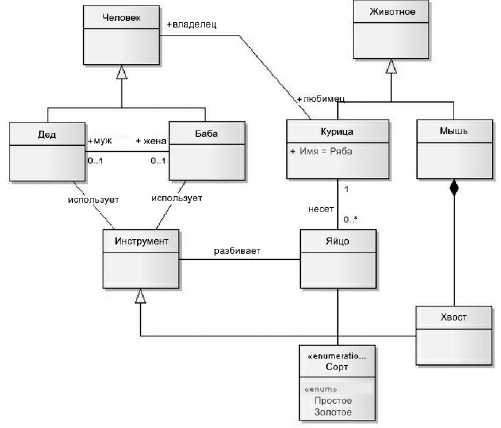


Рисунок 15 –Моделирования отношений между классами на примере сказки

### Выводы

* Инкапсуляция защищает внутреннее устройство объекта и реализуется путем ограничения доступа к атрибутам и операциям класса из других частей программы.
* Обобщение позволяет повторно использовать уже существующие решения, создавая новые классы путем наследования от имеющихся классов.
* Полиморфизм позволяет работать с группой разнородных объектов одинаковым образом, не задумываясь о различиях в реализации.
* Инкапсуляция, наследование и полиморфизм - три кита, на которых держится ООП.
* В любой системе между объектами существуют отношения разных типов.
* Отношение зависимости означает, что реализация одного класса зависит от спецификации операций другого класса.
* Ассоциация выражает отношение между несколькими равноправными объектами и может иметь направление, роли и кратность, а также изображаться в виде класса ассоциации.
* Композиция и агрегация используются, если между объектами существуют отношения типа "часть-целое", причем композиция предполагает, что части не могут существовать отдельно от целого.

**Порядок выполнения работы**

1. Изучить теоретический материал. Выполнить примеры
2. Согласно заданному варианту, выполнить проектирование системы:

* создать иерархию классов системы;
* для классов указать стереотипы;
* разместить классы по пакетам в Designmodel, как это описано в методичке (придерживаться принципа: количество связей внутри пакета больше количества связей, проходящих за границы пакета);
* связать объекты на диаграммах взаимодействия с классами, а сообщения – с операциями;
* каждый класс снабдить описанием, которое должно включать в себя краткое описание (ответственность класса), описание атрибутов в виде таблицы (имя, описание, тип), таблицу с описанием операций (имя, описание, сигнатура);
* построить диаграммы классов системы, отображающие связи между классами;
* для описания поведения экземпляров отдельных классов построить диаграммы состояний;
* построить диаграммы деятельности для моделирования сложных операций (с альтернативами, циклами);

**Контрольные вопросы**

1. Какие три принципа лежат в основе ООП?
2. Что такое интерфейс? На каком из базовых принципов ООП основан *механизм интерфейсов*?
3. Что такое n-арная ассоциация?
4. В чем разница между агрегацией и композицией?
5. Что такое класс ассоциации?

### Лабораторная работа №№12-13

### Тема: Работа с CASE – средствами кодирования программного обеспечения

**Цель работы:** подготовкамодели к генерациипрограммногокода; генерацияпрограммногокода

### Теоретическая часть

Одно из х свойств StarUML – возможность генерации программного кода, представляющего модель. Пакет способен выполнять кодогенерацию на языках C++, C#, Java.

**Подготовка к генерации программного кода**

Процесс генерации программного кода состоит из пяти основных этапов:

* 1. Проверка модели.
  2. Создание компонентов.
  3. Отображение классов на компоненты.
  4. Установка свойств генерации программного кода.
  5. Генерация программного кода.

В разных языках не все этапы обязательны. Так, программы, разработанные на языке С++, генерируются и без предварительного создания компонентов. Генерировать код программ на любом языке можно, не выполняя проверки модели, хотя во время генерации это порой приводит к различным ошибкам.

Хотя не все этапы обязательны, первые пять из них рекомендуется выполнять до начала процесса генерации. Проверка модели поможет выявить ее неточности и недостатки, нежелательные с точки зрения последующей генерации программ. Этапы, на которых рассматриваются компоненты, - это способ отобразить логическую схему системы на ее физическую реализацию, причем на этих этапах собирается большой объем полезной информации. Если пропустить эти этапы, то для создания компонентов Rose воспользуется пакетной структурой логического представления.

**Проверка модели**

В StarUML существует не зависящее от языка средство проверки моделей, применяемое для обеспечения корректности модели перед генерацией программного кода. Такая проверка помогает выявить в модели неточности и ошибки, не позволяющие генерировать программы надлежащим образом.

В общем случае проверка модели может выполняться на любом этапе работы над проектом. Однако после завершения разработки графических диаграмм она является обязательной, поскольку позволяет выявить целый ряд ошибок разработчика. К числу таких ошибок и предупреждений относятся, например, не используемые ассоциации и классы, оставшиеся после удаления отдельных графических элементов с диаграмм, а также операции, не являющиеся именами сообщений на диаграммах взаимодействия.

Для проверки модели следует выполнить операцию главного меню: ModelОписание: srarrVerifyModel. Результаты верификации разработанной модели отображаются в окне (рис. 1):

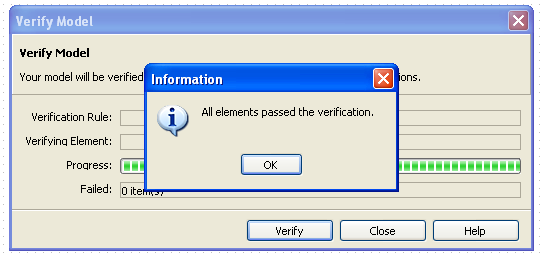


Рисунок 1- Окно верификации

Результаты проверки разработанной модели на наличие ошибок отображаются в окне журнала. Прежде чем приступить к генерации текста программного кода разработчику следует добиться устранения всех ошибок и предупреждений, о чем должно свидетельствовать чистое окно журнала (рис. 2).

Рисунок 2- Вид журнала при отсутствии ошибок по результатам проверки модели

К наиболее распространенным ошибкам относятся, например, сообще­ния на диаграмме Последовательности или Кооперативной диаграмме, не отображенные на операцию, либо объекты этих диаграмм, не отображенные на класс.

**Создание компонентов**

Второй этап процесса генерации программного кода - создание компонентов для классов. Существуют компоненты самых разных типов: файлы исходного программного кода, исполняемые файлы, библиотеки исполнения в реальном времени, компоненты ActiveX, апплеты и т.д. Перед генерацией программ можно отобразить каждый из классов на соответствующий компонент исходного программного кода.

После создания компонентов можно добавить зависимости между ними на диаграмме Компонентов. Зависимости между компонентами - это зависимости во время компиляции системы.

При генерации программ на C++, Java или VisualBasic выполнять подобный шаг не обязательно. В Java и VisualBasicRose создаст автоматически соответствующий компонент для каждого из классов.

Для создания компонента:

1. Откройте диаграмму ComponentDiagram.

2. С помощью значка Component панели инструментов Diagram введите новый компонент в диаграмму.

**Отображение классов на компоненты**

Каждый компонент исходного кода - это файл с исходным программным кодом для одного или нескольких классов. В C++ каждый класс отображается на два компонента с исходным кодом: файл заголовка и основной файл (тело). В PowerBuilder на один компонент отображается несколько классов. Компонентом с исходным программным кодом в PowerBuilder является файл библиотеки PowerBuilder (.pbl). В Java каждый компонент - это один файл .java. Компоненты также создаются для элементов управления ActiveX, апплетов, файлов DDL, исполняемых файлов, а также других исходных и скомпилированных файлов.

Третий этап процесса генерации программного кода - отображение каждого из классов на соответствующие компоненты. В PowerBuilder необходимо отобразить каждый класс на компонент перед генерацией программы, в то время как в C++, Java и VisualBasic этот шаг не является обязательным. StarUML может генерировать программный код самостоятельно. При генерации в StarUML программ Java и VisualBasic производится еще и генерация нужных компонентов и отображение классов. Однако для C++ компоненты не создаются автоматически, а кроме того, ни для одного из языков не генерируются зависимости. Поэтому рекомендуется выполнять этот шаг независимо от применяемого языка программирования. Для отображения класса на компонент:

1. Щелкните правой кнопкой мыши на компоненте, на диаграмме Компонентов или в браузере.

2. Выберите OpenSpecification в контекстном меню.

3. Выберите вкладку Realizes (Реализует).

4. Во вкладке Realizes щелкните правой кнопкой мыши на нужном классе (классах) и выберите Assign (Присвоить) в контекстном меню.

В браузере имя компонента будет показано в круглых скобках вслед за именем класса в Логическом представлении (Logicalview).

ИЛИ

1. Найдите класс в окне Logicalview браузера.

2. Перетащите класс на нужный компонент в окне Componentview.

3. В окне Logicalview имя компонента будет показано в круглых скобках за именем класса.

**Установка свойств генерации программного кода**

Для каждого языка в StarUML предусмотрен ряд определенных свойств гене­рации программного кода. Можно установить несколько параметров генерации программного кода для классов, атрибутов, компонентов и других элементов модели. Этими свойствами определяется способ генерации программ. В StarUML предлагаются общепринятые параметры по умолчанию.

Настройка свойств C++

Для того чтобы пользоваться возможностями C++, необходимо описать назначение этих свойств, список которых доступен во вкладке C++ окна Options (Tools-Option). (см. рис. 3)

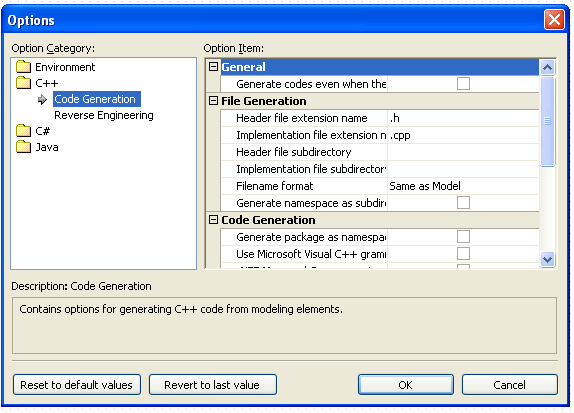


Рисунок 3 – Окно свойств генерации кода на С++

Перед генерацией программного кода рекомендуется анализировать эти свойства и вносить необходимые изменения.

Назначение свойств:

1. **CodeName** - устанавливает имя класса в создаваемом коде. Данное свойство необходимо устанавливать только в том случае, если имя класса должно быть отлично от имени заданного в модели RationalRose. Данное свойство необходимо использовать для создания работоспособного кода C++, если для классов в модели используются русские имена.
2. **ImplementationType** - позволяет использовать простые типы вместо определения класса, устанавливаемого RationalRose по умолчанию. При задании этого параметра создается директива typedef.
3. **ClassKey** - используется для задания типа класса, такого как class, struct, или union. Если тип не указан, то создается класс.
4. **GenerateEmptyRegion** - свойство указывает, как будет создаваться пустой раздел protected: None - пустой раздел не будут создан; Preserved - пустой раздел будет создан, если будет установлено свойство «preserve=yes»; Unpreserved — пустой раздел будет создан, если будет установлено свойство «preserve=no»; All — всегда будет создаваться.
5. **PutBodiesInSpec** - если установлено как true, то в заголовочный файл попадет и описание тела класса. Используется для компиляторов, которым необходимо определение шаблона класса в каждом компилируемом файле.
6. **GenerateDefaultConstructor** - позволяет установить, необходимо ли создавать конструктор для класса по умолчанию. Может принимать следующие значения: DeclareAndDefine - создается определение для конструктора и скелет конструктора в теле класса; DeclareOnly - создается только определение; DoNotDeclare - не создается ни определения, ни скелета конструктора.
7. **DefaultConstructorVisibility** - устанавливает раздел, в котором будет определен конструктор по умолчанию: public, protected, private, implementation.
8. **InlineDefaultConstructor** - устанавливает, будет ли конструктор по умолчанию создаваться как inline подстановка. Если конструктора по умолчанию нет, то данное свойство не оказывает на код никакого эффекта.
9. **ExplicitDefaultConstructor** - устанавливает конструктор по умолчанию как explicit (явно заданный).
10. **InlineRelationalOperations** - определяет, будут ли функции операторов сравнения создаваться как inline подстановка.
11. **GenerateStorageMgmtOperations** - определяет, будут ли переопределяться операторы new и delete в классе.
12. **StorageMgmtVisibility** - определяет раздел, в который будут помещены операторы new и delete.
13. **InlineStorageMgmtOperations** - определяет, будут ли операторы new и delete определены как inline подстановка.
14. **GenerateSubscriptOperation** - определяет, будет ли переопределен оператор [].
15. **SubscriptVisibility**определяет - раздел, в который будет помещен оператор [].
16. **SubscriptKind** - определяет вид функций оператора []: Common - обычная, Virtual - виртуальная, Abstract - абстрактная.
17. **SubscriptResultType** - определяет тип возвращаемого выражения для

оператора [].

1. **InlineSubscriptOperation** - определяет, будет ли оператор [] определен как inline подстановка.
2. **GenerateDereferenceOperation** - определяет, будет ли переопределен оператор \*.
3. **DereferenceVisibility** - определяет раздел, в который будет помещен оператор \*.
4. **DereferenceKind** - определяет вид функций оператора \*: Common - обычная, Virtual - виртуальная, Abstract - абстрактная.
5. **DereferenceResultType** - определяет тип возвращаемого выражения для оператора \*.
6. **InlineDereferenceOperation** - определяет, будет ли оператор \* определен, как inline подстановка.
7. **GeneratelndirectionOperation** - определяет, будет ли переопределен оператор ->.
8. **IndirectionVisibility** - определяет раздел, в который будет помещен оператор ->.
9. **IndirectionKind** - определяет вид функций оператора ->: Common - обычная, Virtual - виртуальная, Abstract - абстрактная.
10. **IndirectionResultType** - определяет тип возвращаемого выражения для оператора ->.
11. **InlinelndirectionOperation** - определяет, будет ли оператор -> определен как inline подстановка.
12. **GenerateStreamOperations** - определяет, будут ли переопределены операторы потоков (« и »).

**Выбор класса, компонента или пакета**

При генерации программного кода за один раз можно создать класс, компонент или целый пакет. Программный код генерируется с помощью диаграммы или браузера. При генерации кода из пакета можно выбрать либо пакет Логического представления на диаграмме Классов, либо пакет представления Компонентов на диаграмме Компонентов. При выборе пакета Логического представления генерируются все его классы. При выборе пакета представления Компонентов генерируются все его компоненты.

Программный код можно генерировать одновременно для нескольких классов, компонентов или пакетов. На диаграмме с помощью клавиши Ctrl выберите классы, компоненты или пакеты, для которых нужно сгенерировать программный код, а затем - соответствующую команду генерации в меню.

**Генерация программного кода**

D меню Tools предлагается несколько вариантов генерации, специфичных для конкретного языка программирования (см. рис. 4).

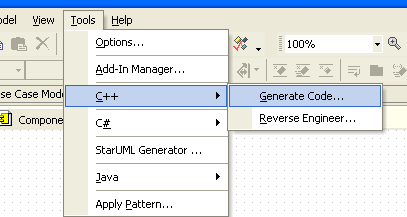


Рисунок 4 - Пункты меню генерации кода

Чтобы показать или скрыть эти пункты меню, выберите пункт Tools → Add-InManager (Надстройки → Менеджер надстроек). В диалоговом окне Add-InManager (см. рис. 5) с помощью флажков покажите или скройте нужные варианты для различных языков.

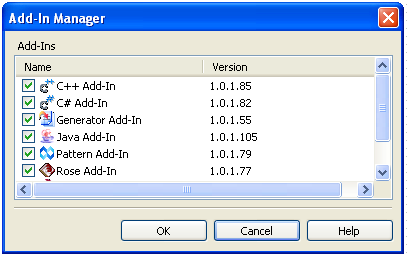


Рисунок 5 - Менеджер надстроек Add-InManager

Генерация программного кода в среде StarUML возможна для отдельного класса или компонента. Для этого нужный элемент модели предварительно следует выделить в браузере проекта и выполнить операцию контекстного меню: Tools→C++→CodeGeneration. В результате этого будет открыто диалоговое окно с предложением выбора классов для генерации программного кода на выбранном языке программирования (рис. 6).

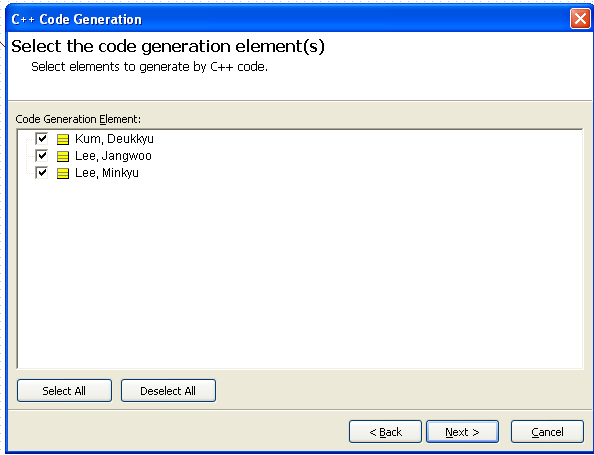


Рисунок 6 - Окно выбора классов для генерации программного кода

После выбора соответствующих классов и нажатия кнопки OK и настройки параметров на следующем шаге (рисунок 7) программа StarUML выполняет кодогенерацию.

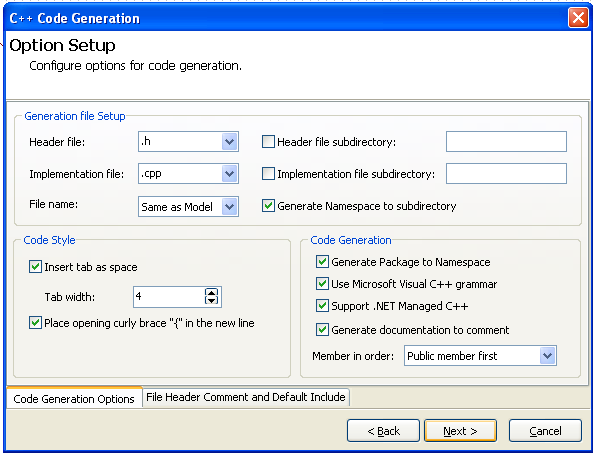


Рисунок 7 - Окно настройки параметров для генерации программного кода

Затем происходит компиляция и выдается окно статуса CodeGeneration. Здесь можно увидеть информацию о том, какой класс был закодирован и количество ошибок и предупреждений (рис. 8). Если у вас произошла, какая-либо ошибка или же предупреждение, то их можно увидеть на рабочем поле в StarUML, для этого и существует самое нижнее окно, в нем передаются все ваши действия и ошибки, произошедшие в ходе кодогенерации.

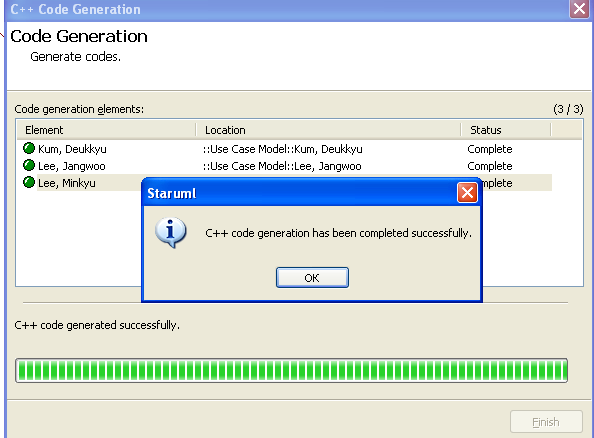


Рисунок 8 – Окно статуса компиляции

**Результаты генерации**

В результате кодогенерацииStarUML создает два файла с расширением “.h” и “.cpp”, названия у них те же, что и название класса. Итак, выполнив эти действия, нажимаем правой клавишей на класс, появляется окошко, в нем ищем “С++”, и видим два пункта BrowseHeader и BrowseBody, и в зависимости от того какой из файлов нам нужен “.h” (заголовочный) или “.cpp” (непосредственно реализация), выбираем их. Эти файлы открываются с помощью блокнота и теперь легко можно увидеть скелет класса, с различными комментариями, которые писали вы на диаграммах, и комментарии которые вставляет сама StarUML. Теперь можно открыть один из файлов в С++ и доработать класс, описать работу функций, добавить различные нововведения.

Следует заметить, что при установленной на компьютер разработчика интегрированной среды сгенерированные файлы с текстом программного кода автоматически открываются в этой среде после двойного щелчка на пиктограмме этих файлов. Тем не менее, лучше копировать содержимое этих файлов в предварительно созданные программные проекты для полного контроля в этих средах процесса программирования и отладки приложений.

Сгенерированные программой StarUML файлы с текстом программного кода содержат минимум информации. Для включения дополнительных элементов в программный код следует изменить свойства генерации программного кода, установленные по умолчанию.

В заключение следует отметить, что эффект от использования средства StarUML проявляется при разработке *масштабных проектов* в составе команды или проектной группы. Однако ситуация покажется не столь тривиальной, когда станет необходимо выполнить проект с несколькими десятками вариантов использования и сотней *классов*. Именно для подобных проектов явно выявляется преимущество использования средства StarUML 3 и нотации языка UML для документирования и реализации соответствующих моделей.

**Порядок выполнения работы**

1. Изучитьтеоретическийматериал
2. Сгенерировать программный код на С++ для диаграммы классов, разработанной вами в предыдущих лабораторных работах.

**Содержание отчета**

1. Титульный лист
2. Постановка задачи
3. Листинг сгенерированного кода
4. Вывод

**Контрольные вопросы**

1. Охарактеризовать основные этапыпроцесса генерации программного кода.
2. Какие диаграммы необходимо предварительно разработать, чтобы выполнить кодогенерацию?
3. Как посмотреть исходный код?
4. Какие установки свойств доступны на вкладке C++?
5. Какова структура создаваемого кода?
6. Что необходимо добавить в шаблоны классов для получения работоспособного приложения?
7. Какие шаги нужно предпринять для обновления модели по исходному коду?
8. Какие основные этапы кодогенерации вы знаете? Расскажите кратко о каждом из них?

### Лабораторная работа № 14

### Тема: Работа с CASE – средствами тестирования программного обеспечения

**Цель работы:** отладка разработанного программного средства; тестирование разработанного программного средства.

### Теоретическая часть

*Отладка* ПС − это деятельность, направленная на обнаружение и исправление ошибок в ПС с использованием процессов выполнения его программ. *Тестирование* ПС − это процесс выполнения его программ на некотором наборе данных, для которого заранее известен результат применения или известны правила поведения этих программ. Указанный набор данных называется тестовым или просто тестом. Таким образом, отладку можно представить в виде многократного повторения трех процессов: тестирования, в результате которого может быть констатировано наличие в ПС ошибки, поиска места ошибки в программах и документации ПС и редактирования программ и документации с целью устранения обнаруженной ошибки.

Отладка программного средства невозможна без представления физической структуры. На этом этапе проектирования необходимо разработать *диаграмму компонентов*.

*Диаграммой компонентов* (Componentdiagram) называется диаграмма UML, на которой показаны компоненты системы и зависимости между ними.

*Компонентом* называется физический модуль кода. Компонентами бывают как библиотеки исходного кода, так и исполняемые файлы. Например, .h и .cpp и .exe - будут отдельными компонентами.

### Диаграмма компонентов и особенности ее построения

Все рассмотренные ранее диаграммы отражали концептуальные и логические аспекты построения модели системы. Особенность логического представления заключается в том, что оно оперирует понятиями, которые не имеют материального воплощения. Другими словами, различные элементы логического представления, такие как классы, ассоциации, состояния, сообщения, не существуют материально или физически. Они лишь отражают понимание статической структуры той или иной системы или динамические аспекты ее поведения.

Для создания конкретной физической системы необходимо реализовать все элементы логического представления в конкретные материальные сущности. Для описания таких реальных сущностей предназначен другой аспект модельного представления, а именно – физическое представление модели. В контексте языка UML это означает совокупность связанных физических сущностей, включая программное и аппаратное обеспечение, а также персонал, которые организованы для выполнения специальных задач.

Физическая система (physicalsystem) — реально существующий прототип модели системы.

С тем чтобы пояснить отличие логического и физического представлений, необходимо в общих чертах рассмотреть процесс разработки программной системы. Ее исходным логическим представлением могут служить структурные схемы алгоритмов и процедур, описания интерфейсов и концептуальные схемы баз данных. Однако для реализации этой системы необходимо разработать исходный текст программы на языке программирования. При этом уже в тексте программы предполагается организация программного кода, определяемая синтаксисом языка программирования и предполагающая разбиение исходного кода на отдельные модули.

Однако исходные тексты программы еще не являются окончательной реализацией проекта, хотя и служат фрагментом его физического представления. Программная система может считаться реализованной в том случае, когда она будет способна выполнять функции своего целевого предназначения. А это возможно, только если программный код системы будет реализован в форме исполняемых модулей, библиотек классов и процедур, стандартных графических интерфейсов, файлов баз данных. Именно эти компоненты являются базовыми элементами физического представления системы в нотации языка UML.

Полный проект программной системы представляет собой совокупность моделей логического и физического представлений, которые должны быть согласованы между собой. В языке UML для физического представления моделей систем используются так называемые диаграммы реализации, которые включают в себя две отдельные канонические диаграммы: диаграмму компонентов и диаграмму развертывания.

Диаграмма компонентов, в отличие от ранее рассмотренных диаграмм, описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код. Во многих средах разработки модуль или компонент соответствует файлу. Пунктирные стрелки, соединяющие модули, показывают отношения взаимозависимости, аналогичные тем, которые имеют место при компиляции исходных текстов программ. Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними.

В разработке диаграмм компонентов участвуют как системные аналитики и архитекторы, так и программисты. Диаграмма компонентов обеспечивает согласованный переход от логического представления к конкретной реализации проекта в форме программного кода. Одни компоненты могут существовать только на этапе компиляции программного кода, другие – на этапе его исполнения. Диаграмма компонентов отражает общие зависимости между компонентами, рассматривая последние в качестве отношений между ними.

### Компоненты

Для представления физических сущностей в языке UML применяется специальный термин – компонент.

Компонент (component) — физически существующая часть системы, которая обеспечивает реализацию классов и отношений, а также функционального поведения моделируемой программной системы.

Компонент предназначен для представления физической организации ассоциированных с ним элементов модели. Дополнительно компонент может иметь текстовый стереотип и помеченные значения, а некоторые компоненты – собственное графическое представление. Компонентом может быть исполняемый код отдельного модуля, командные файлы или файлы, содержащие интерпретируемые скрипты.

Компонент служит для общего обозначения элементов физического представления модели и может реализовывать некоторый набор интерфейсов. Для графического представления компонента используется специальный символ – прямоугольник со вставленными слева двумя более мелкими прямоугольниками (рис. 1). Внутри объемлющего прямоугольника записывается имя компонента и, возможно, дополнительная информация. Этот символ является базовым обозначением компонента в языке UML.

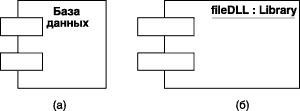


Рисунок 1 - Графическое изображение компонента

Графическое изображение компонента ведет свое происхождение от обозначения модуля программы, применявшегося некоторое время для отображения особенностей инкапсуляции данных и процедур.

Модуль (module) — часть программной системы, требующая памяти для своего хранения и процессора для исполнения.

В этом случае верхний маленький прямоугольник концептуально ассоциировался с данными, которые реализует этот компонент (иногда он изображается в форме овала). Нижний маленький прямоугольник ассоциировался с операциями или методами, реализуемыми компонентом. В простых случаях имена данных и методов записывались явно в маленьких прямоугольниках, однако в языке UML они не указываются.

Имя компонента подчиняется общим правилам именования элементов модели в языке UML и может состоять из любого числа букв, цифр и знаков препинания. Отдельный компонент может быть представлен на уровне типа или экземпляра. И хотя его графическое изображение в обоих случаях одинаково, правила записи имени компонента несколько отличаются.

Если компонент представляется на уровне типа, то записывается только имя типа с заглавной буквы в форме: <Имя типа>. Если же компонент представляется на уровне экземпляра, то его имя записывается в форме: <имя компонента ‘:' Имя типа>. При этом вся строка имени подчеркивается. Так, в первом случае (рис. 1, а) для компонента уровня типов указывается имя типа, а во втором (рис. 1, б) для компонента уровня экземпляра – собственное имя компонента и имя типа.

Правила именования объектов в языке UML требуют подчеркивания имени отдельных экземпляров, но применительно к компонентам подчеркивание их имени часто опускают. В этом случае запись имени компонента со строчной буквы характеризует компонент уровня примеров.

В качестве собственных имен компонентов принято использовать имена исполняемых файлов, динамических библиотек, Web-страниц, текстовых файлов или файлов справки, файлов баз данных или файлов с исходными текстами программ, файлов скриптов и другие.

В отдельных случаях к простому имени компонента может быть добавлена информация об имени объемлющего пакета и о конкретной версии реализации данного компонента. Необходимо заметить, что в этом случае номер версии записывается как помеченное значение в фигурных скобках. В других случаях символ компонента может быть разделен на секции, чтобы явно указать имена реализованных в нем классов или интерфейсов. Такое обозначение компонента называется **расширенным**.

Поскольку компонент как элемент модели может иметь различную физическую реализацию, иногда его изображают в форме специального графического символа, иллюстрирующего конкретные особенности реализации. Строго говоря, эти дополнительные обозначения не специфицированы в нотации языка UML. Однако, удовлетворяя общим механизмам расширения языка UML, они упрощают понимание диаграммы компонентов, существенно повышая наглядность графического представления.

Для более наглядного изображения компонентов были предложены и стали общепринятыми следующие графические стереотипы:

* Во-первых, стереотипы для компонентов развертывания, которые обеспечивают непосредственное выполнение системой своих функций. Такими компонентами могут быть динамически подключаемые библиотеки (рис. 2, а), Web-страницы на языке разметки гипертекста (рис. 2, б) и файлы справки (рис. 2, в).
* Во-вторых, стереотипы для компонентов в форме рабочих продуктов. Как правило – это файлы с исходными текстами программ (рис. 2, г).

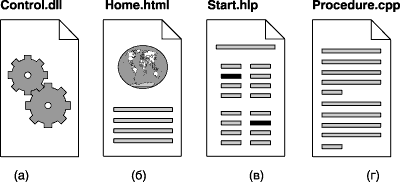


Рисунок 2 - Варианты графического изображения компонентов на диаграмме компонентов

Эти элементы иногда называют **артефактами** , подчеркивая при этом их законченное информационное содержание, зависящее от конкретной технологии реализации соответствующих компонентов. Более того, разработчики могут для этой цели использовать самостоятельные обозначения, поскольку в языке UML нет строгой нотации для графического представления артефактов.

Другой способ спецификации различных видов компонентов — указание текстового стереотипа компонента перед его именем. В языке UML для компонентов определены следующие стереотипы:

* <<file>> (файл) – определяет наиболее общую разновидность компонента, который представляется в виде произвольного физического файла.
* <<executable>> (исполнимый) – определяет разновидность компонента-файла, который является исполнимым файлом и может выполняться на компьютерной платформе.
* <<document>> (документ) – определяет разновидность компонента-файла, который представляется в форме документа произвольного содержания, не являющегося исполнимым файлом или файлом с исходным текстом программы.
* <<library>> (библиотека) – определяет разновидность компонента-файла, который представляется в форме динамической или статической библиотеки.
* <<source>> (источник) – определяет разновидность компонента-файла, представляющего собой файл с исходным текстом программы, который после компиляции может быть преобразован в исполнимый файл.
* <<table>> (таблица) – определяет разновидность компонента, который представляется в форме таблицы базы данных.

Отдельными разработчиками предлагались собственные графические стереотипы для изображения тех или иных типов компонентов, однако, за небольшим исключением они не нашли широкого применения. В свою очередь ряд инструментальных CASE-средств также содержат дополнительный набор графических стереотипов для обозначения компонентов.

### Интерфейсы

Следующим графическим элементом диаграммы компонентов являются интерфейсы. В общем случае интерфейс графически изображается окружностью, которая соединяется с компонентом отрезком линии без стрелок (рис. 3, а). При этом имя интерфейса, которое рекомендуется начинать с заглавной буквы "I", записывается рядом с окружностью. Семантически линия означает реализацию интерфейса, а наличие интерфейсов у компонента означает, что данный компонент реализует соответствующий набор интерфейсов .

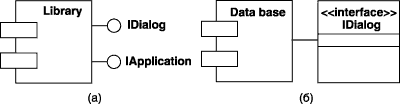


Рисунок 3 - Графическое изображение интерфейсов на диаграмме компонентов.

Кроме того, интерфейс на диаграмме компонентов может быть изображен в виде прямоугольника класса со стереотипом <<interface>> и секцией поддерживаемых операций (рис. 3, б). Как правило, этот вариант обозначения используется для представления внутренней структуры интерфейса.

При разработке программных систем интерфейсы обеспечивают не только совместимость различных версий, но и возможность вносить существенные изменения в одни части программы, не изменяя другие . Характер применения интерфейсов отдельными компонентами может отличаться.

Различают два способа связи интерфейса и компонента. Если компонент реализует некоторый интерфейс, то такой интерфейс называют экспортируемым или **поддерживаемым**, поскольку этот компонент предоставляет его в качестве сервиса другим компонентам. Если же компонент использует некоторый интерфейс, который реализуется другим компонентом, то такой интерфейс для первого компонента называется импортируемым. Особенность импортируемого интерфейса состоит в том, что на диаграмме компонентов это отношение изображается с помощью зависимости.

### Зависимости между компонентами

В общем случае отношение зависимости также было рассмотрено ранее. Отношение зависимости служит для представления факта наличия специальной формы связи между двумя элементами модели, когда изменение одного элемента модели оказывает влияние или приводит к изменению другого элемента модели. Отношение зависимости на диаграмме компонентов изображается пунктирной линией со стрелкой, направленной от клиента или зависимого элемента к источнику или независимому элементу модели.

Зависимости могут отражать связи отдельных файлов программной системы на этапе компиляции и генерации объектного кода. В других случаях зависимость может указывать на наличие в независимом компоненте описаний классов, которые используются в зависимом компоненте для создания соответствующих объектов. Применительно к диаграмме компонентов зависимости могут связывать компоненты и импортируемые этим компонентоминтерфейсы, а также различные виды компонентов между собой.

В этом случае рисуют стрелку от компонента-клиента к импортируемому интерфейсу (рис. 4). Наличие такой стрелки означает, что компонент не реализует соответствующий интерфейс, а использует его в процессе своего выполнения. При этом на этой же диаграмме может присутствовать и другой компонент, который реализует этот интерфейс. Отношение реализации интерфейса обозначается на диаграмме компонентов обычной линией без стрелки.

Так, например, изображенный ниже фрагмент диаграммы компонентов представляет информацию о том, что компонент с именем Control зависит от импортируемого интерфейсаIDialog, который, в свою очередь, реализуется компонентом с именем DataBase . При этом для второго компонентa этот интерфейс является экспортируемым. Изобразить связь второго компонентaDataBase с этим интерфейсом в форме зависимости нельзя , поскольку этот компонент реализует указанный интерфейс.

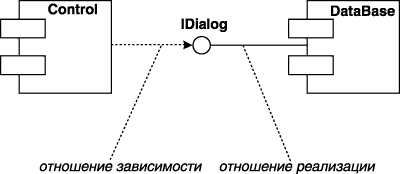


Рисунок 4 –Фрагмент диаграммы компонентов

Другим случаем отношения зависимости на диаграмме компонентов является отношение программного вызова и компиляции между различными видами компонентов. Для рассмотренного фрагмента диаграммы компонентов (рис. 5) наличие подобной зависимости означает, что исполнимый компонентControl .exe использует или импортирует некоторую функциональность компонентaLibrary .dll, вызывает страницу гипертекста Home .html и файл помощи Search .hlp, а исходный текст этого исполнимого компонентa хранится в файле Control .cpp. При этом характер отдельных видов зависимостей может быть отмечен дополнительно с помощью текстовых стереотипов.

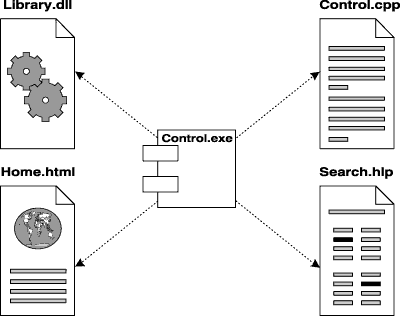


Рисунок 5 - Графическое изображение отношения зависимости между компонентами

На диаграмме компонентов могут быть также представлены отношения зависимости между компонентами и реализованными в них классами. Эта информация имеет значение для обеспечения согласования логического и физического представлений модели системы. Разумеется, изменения в структуре описаний классов могут привести к изменению этой зависимости. Ниже приводится фрагмент зависимости подобного рода, когда исполнимый компонентControl .exe зависит от соответствующих классов (рис. 6).

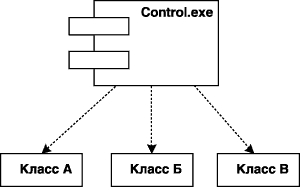


Рисунок 6 - Графическое изображение зависимости между компонентом и классами

В этом случае из диаграммы компонентов не следует, что классы реализованы данным компонентом. Если требуется подчеркнуть, что некоторый компонент реализует отдельные классы, то для обозначения компонентa используется расширенный символ прямоугольника. При этом прямоугольник компонентa делится на две секции горизонтальной линией. Верхняя секция служит для записи имени компонентa и, возможно, дополнительной информации, а нижняя секция – для указания реализуемых данным компонентом классов (рис. 7).

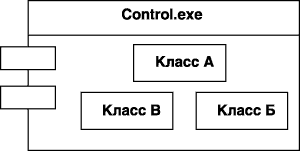


Рисунок 7 - Графическое изображение компонентa с информацией о реализуемых им классах

В случае если компонент является экземпляром и реализует три отдельных объекта, он изображается в форме компонентa уровня экземпляров (рис. 8). Объекты, которые находятся в отдельном компоненте-экземпляре, изображаются вложенными в символ данного компонента. Подобная вложенность означает, что выполнение компонентa влечет за собой выполнение операций соответствующих объектов. При этом существование компонентa в течение времени исполнения программы обеспечивает функциональность всех вложенных в него объектов. Что касается доступа к этим объектам, то он может быть дополнительно специфицирован с помощью видимости, подобно видимости пакетов.

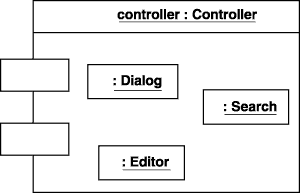


Рисунок 8 - Графическое изображение компонента-экземпляра, реализующего отдельные объекты

Для компонентов с исходным текстом программы видимость может означать возможность внесения изменений в соответствующие тексты программ с их последующей перекомпиляцией. Для компонентов с исполняемым кодом программы видимость может характеризовать возможность запуска на исполнение соответствующего компонентa или вызова реализованных в нем операций или методов.

### Рекомендации по построению диаграммы компонентов

Разработка диаграммы компонентов предполагает использование информации не только о логическом представлении модели системы, но и об особенностях ее физической реализации. В первую очередь, необходимо решить, из каких физических частей или файлов будет состоять программная система. На этом этапе следует обратить внимание на такую реализацию системы, которая обеспечивала бы возможность повторного использования кода за счет рациональной декомпозиции компонентов, а также создание объектов только при их необходимости.

Общая производительность программной системы существенно зависит от рационального использования вычислительных ресурсов. Для этой цели необходимо большую часть описаний классов, их операций и методов вынести в динамические библиотеки, оставив в исполняемых компонентах только самые необходимые для инициализации программы фрагменты программного кода.

После общей структуризации физического представления системы необходимо дополнить модель интерфейсами и схемами базы данных. При разработке интерфейсов следует обращать внимание на согласование различных частей программной системы. Включение в модель схемы базы данных предполагает спецификацию отдельных таблиц и установление информационных связей между ними.

Завершающий этап построения диаграммы компонентов связан с установлением и нанесением на диаграмму взаимосвязей между компонентами, а также отношений реализации. Эти отношения должны иллюстрировать все важнейшие аспекты физической реализации системы, начиная с особенностей компиляции исходных текстов программ и заканчивая исполнением отдельных частей программы на этапе ее выполнения. Для этой цели можно использовать различные графические стереотипы компонентов.

При разработке диаграммы компонентов следует придерживаться общих принципов создания моделей на языке UML. В частности, в первую очередь необходимо использовать уже имеющиеся в языке UML и общепринятые графические и текстовые стереотипы. В большинстве типовых проектов этого набора достаточно для представления компонентов и зависимостей между ними.

Если же проект содержит физические элементы, описание которых отсутствует в языке UML, то следует воспользоваться механизмом расширения. В частности, можно применить дополнительные стереотипы для отдельных нетиповых компонентов или помеченные значения для уточнения отдельных характеристик компонентов.

**Особенности разработки диаграмм компонентов в среде StarUML**

Диаграмма *компонентов* служит частью физического представления модели, играет важную роль в процессе ООАП и является необходимой для генерации программного кода. Для разработки диаграмм *компонентов* в браузере проекта предназначено отдельное представление *компонентов* (ComponentView), в котором уже содержится диаграмма *компонентов* с пустым содержанием и именем по умолчанию Main (Главная).

Активизация диаграммы *компонентов* может быть выполнена одним из следующих способов:

Щелкнуть на кнопке с изображением диаграммы *компонентов* на стандартной панели инструментов.

Раскрыть представление *компонентов* в обозревателе моделей и дважды щелкнуть на пиктограмме Main (Главная).

Черезпунктменю Add DiagramОписание: srarrComponent Diagram.

В результате выполнения этих действий появляется новое окно с чистым рабочим листом диаграммы *компонентов* и специальная панель инструментов, содержащая кнопки с изображением графических примитивов, необходимых для разработки диаграммы *компонентов* (табл.1).

Таблица 1 - Назначение кнопок специальной панели инструментов диаграммы компонентов

|  |  |  |
| --- | --- | --- |
| **Графическое изображение** | **Всплывающая подсказка** | **Назначение кнопки** |
| Описание: 12_1 | SelectionTool | Превращает изображение курсора в форму стрелки для последующего выделения элементов на диаграмме |
| Описание: 12_2 | TextBox | Добавляет на диаграмму текстовую область |
| Описание: 12_3 | Note | Добавляет на диаграмму примечание |
| Описание: 12_4 | AnchorNotetoItem | Добавляет на диаграмму связь примечания с соответствующим графическим элементом диаграммы |
| Описание: 12_5 | Component | Добавляет на диаграмму компонент |
| Описание: 12_6 | Package | Добавляет на диаграмму пакет |
| Описание: 12_7 | Dependency | Добавляет на диаграмму отношение зависимости |
| Описание: 12_8 | SubprogramSpecification | Добавляет на диаграмму спецификацию подпрограммы |
| Описание: 12_9 | SubprogramBody | Добавляет на диаграмму тело подпрограммы |
| Описание: 12_10 | MainProgram | Добавляет на диаграмму главную программу |
| Описание: 12_11 | PackageSpecification | Добавляет на диаграмму спецификацию пакета |
| Описание: 12_12 | PackageBody | Добавляет на диаграмму тело пакета |
| Описание: 12_13 | TaskSpecification | Добавляет на диаграмму спецификацию задачи |
| Описание: 12_14 | TaskBody | Добавляет на диаграмму тело задачи |
| Описание: 12_15 | GenericSubprogram | Добавляет на диаграмму типовую подпрограммы(по умолчанию отсутствует) |
| Описание: 12_16 | GenericPackage | Добавляет на диаграмму типовой пакет (по умолчанию отсутствует) |
| Описание: 12_17 | Database | Добавляет на диаграмму базу данных (по умолчанию отсутствует) |

Программа StarUML не поддерживает следующие графические стереотипы. Графическое изображение этих стереотипов и их краткая характеристика приводятся в следующей таблице (табл. 2). При этом каждому из компонентов, как правило, соответствует отдельный файл исходной сборки программного приложения.

Использование *стереотипов* существенно увеличивают наглядность графического представления диаграммы *компонентов* и позволяют архитектору уточнить характер реализации модели программистом на выбранном языке программирования.

Таблица 2. Графическое изображение стереотипов компонентов и их характеристика

|  |  |  |
| --- | --- | --- |
| **Графическое изображение и имя по умолчанию** | **Название стереотипа** | **Характеристика стереотипа компонента** |
| Описание: 12_18 | SubprogramSpecification | Спецификация подпрограммы. Содержит описание переменных, процедур и функций и не содержит определений классов |
| Описание: 12_19 | SubprogramBody | Тело подпрограммы. Содержит реализацию процедур и функций, не относящихся к каким-то классам, при этом не содержит определений классов или реализаций операций других классов |
| Описание: 12_20 | MainProgram | Главная программа. Реализует базовую логику работы программного приложения и содержит ссылки на другие компоненты модели |
| Описание: 12_21 | PackageSpecification | Спецификация пакета. Содержит определение класса, его атрибутов и операций. В языке программирования С++ спецификации пакета соответствует отдельный файл с расширением «h» |
| Описание: 12_22 | PackageBody | Тело пакета. Содержит код реализации операций класса. В языке программирования С++ спецификации пакета соответствует отдельный файл с расширением «cpp» |
| Описание: 12_23 | TaskSpecification | Спецификация задачи. Может содержать определение класса, его атрибутов и операций, которые предполагается использовать в независимом потоке управления |
| Описание: 12_24 | TaskBody | Тело задачи. Может содержать реализацию операций класса, которые имеют независимый поток управления. |
| Описание: 12_25 | GenericSubprogram | Типовая подпрограмма. Содержит описание переменных, процедур и функций, которые могут быть использованы в нескольких программных приложениях. При этом типовая подпрограмма не содержит определений классов |
| Описание: 12_26 | GenericPackage | Типовой пакет. Содержит определение класса, его атрибутов и операций, которое может быть использовано в нескольких программных приложениях |
| Описание: 12_27 | Database | База данных. Содержит определение одного или нескольких классов, их атрибутов и, возможно, операций. При этом соответствующие классы могут быть реализованы в форме одной или нескольких таблиц базы данных |

**Добавление компонента на диаграмму компонентов и редактирование его свойств**

Для добавления компонента на диаграмму компонентов нужно с помощью левой кнопки мыши нажать кнопку с изображением пиктограммы компонента на специальной панели инструментов, отпустить левую кнопку мыши и щелкнуть левой кнопкой мыши на свободном месте рабочего листа диаграммы. В результате этих действий на диаграмме появится изображение компонента с маркерами изменения его геометрических размеров и предложенным средой именем по умолчанию, которое разработчику следует изменить (рис. 9).

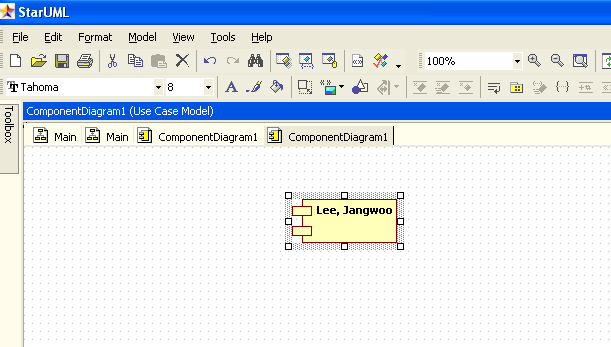
****

Рисунок 9 - Диаграмма компонентов после добавления компонента Main.exe

Для каждого компонента можно определить различные свойства, такие как стереотип, язык программирования, декларации, реализуемые классы. Редактирование этих свойств для произвольного компонента осуществляется с помощью диалогового окна спецификации свойств (рис. 10).

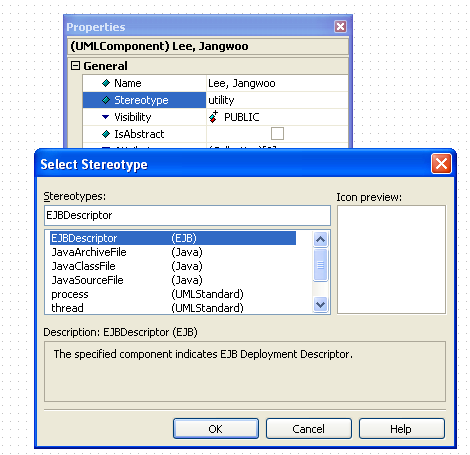


Рисунок 10 - Диалоговоеокно Properties – Select Stereotype

**Добавление отношения зависимости и редактирование его свойств**

Добавление отношения зависимости на диаграмму компонентов аналогично добавлению соответствующего отношения на диаграмму вариантов использования. Для добавления зависимости между двумя компонентами нужно с помощью левой кнопки мыши нажать кнопку с изображением зависимости на специальной панели инструментов, отпустить левую кнопку мыши, щелкнуть левой кнопкой мыши на изображении исходного компонента на диаграмме и отпустить ее на изображении целевого компонента. В результате этих действий на диаграмме появится изображение отношения зависимости в форме пунктирной линии со стрелкой, соединяющей два выбранных компонента.

**Пример построения диаграммы компонентов**

Ниже представлена диаграмма компонентов для программного средства, реализующего алгоритм дискриминантного анализа (рис. 11).



Рисунок 11 – Диаграмма компонентов рассматриваемого ПС

**Тестирование программного средства**

Тестирование – процесс многократного повторения программы с целью обнаружения ошибок. Существуют следующие методы тестирования ПС:

- статическое тестирование (ручная проверка программы за столом);

- детерминированное тестирование (при различных комбинациях исходных данных);

- стохастическое (исходные данные выбираются произвольно, на выходе определяется качественное совпадение результатов или примерная оценка).

При тестировании разработанного ПС необходимо использовать подходящий по функциональности пакет прикладного математического программного обеспечения.

В выбранной среде необходимо произвести расчеты всех параметров, реализованных в ПС, затем следует сравнить результаты и сделать вывод о качестве данного программного продукта.

**Порядок выполнения работы**

1. Изучитьтеоретическийматериал
2. Выполнить тестирование и отладку информационной системы, разработанной в лабораторной работе №№12-13, то есть разработать диаграмму компонентов рассматриваемого ПС

**Содержание отчета**

1. Титульный лист
2. Постановка задачи
3. Тестовый набор данных и результаты тестирования;
4. Вывод

**Контрольные вопросы**

1. Что такое отладка ПС?
2. Что такое тестирование ПС? Какие существуют методы тестирования?
3. Для чего предназначена диаграмма компонентов?
4. Что такое компонент?
5. Как связать класс с компонентом?

### Приложение 1

### Варианты заданий

**1. Автоматизированная информационная система «Индивидуальный план преподавателя»**

**Описание предметной области**.

Для каждого преподавателя (ФИО, Год рождения, Домашний адрес, Контактные телефоны) высшего учебного заведения (Код, Название, Краткое название) на каждый учебный год (Год начала учебного года, Год окончания учебного года) формируется индивидуальный план. В индивидуальном плане отражается общий объем работ преподавателя, который он должен выполнить в течение учебного года. Учет работ ведется по следующей форме:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Наименование работы | План | | Факт | |
| Осенний семестр | Весенний семестр | Осенний семестр | Весенний семестр |
|  |  |  |  |  |  |

В течение учебного года преподаватель выполняет следующие виды работ (Код, Название Краткое название): «Учебная работа», «Учебно-методическая работа», «Научно-методическая работа», «Научно-исследовательская работа», «Организационно-методическая работа», «Внеучебная работа со студентами», «Прочие виды работ». Необходимо вести учет в часах (целых и долях часов) объем запланированных и фактически выполненных объемов работ для каждого преподавателя по семестрам. Для каждого преподавателя также необходимо фиксировать место работы – факультет (Код, Название, Краткое название), кафедра (Код, Название, Краткое название), занимаемую должность (Код, Название, Краткое название), время работы в этой должности (Дата начала, Дата окончания, Ставка, Дата избрания на должность), кем является преподаватель – штатным сотрудником или совместителем. Также для преподавателя фиксируются:

- ученая степень (Код, Название, Краткое название) – доктор, кандидат; каких наук (Код, Название, Краткое название) – технических, экономических и т.п.; год присуждения;

- ученое звание (Код, Название, Краткое название) – профессор, доцент, с.н.с. и т.п.; год присуждения звания.

**Необходимо осуществлять следующую обработку данных:**

- формирование для каждого преподавателя итоговой суммы (в часах) запланированных и выполненных объемов работ по семестрам;

- список преподавателей, у которых фактическое значение выполненных работ превышает плановое (факультет, кафедра, ФИО, уч.степень, уч.звание, должность, семестр, кол-во перевыполненных объемов работ);

- список преподавателей заданной кафедры, имеющих заданную ученую степень на заданную дату.

**2. Автоматизированная информационная система «Обслуживание заказов клиентов»**

**Описание предметной области.**

Предприятие (Код, Название, Краткое название) осуществляет доставку разных товаров (Код, Название, Краткое название) населению. Прием заказов от населения осуществляет специальная служба (Код, Название, Краткое название) предприятия.

Для того чтобы стать потребителем услуг предприятия каждый абонент должен зарегистрироваться, при этом фиксируются его ФИО, адрес, телефон и паспортные данные (Серия, Номер, Дата выдачи, Кем выдан). Каждый абонент в течение дня может сделать несколько заказов (Дата, Время), заказу присваивается номер.

В каждом заказе может содержаться несколько товаров, для каждого указывается количество товара, единица измерения (Код, Название, Краткое Название), цена за единицу товара, общая стоимость товара. Заказ также имеет итоговую сумму. При формировании бланка заказа, который будет подписан абонентом при получении товара фиксируется, оплачен заказ, или абонент получает товар в кредит. Также на бланке заказа указывается: реквизиты предприятия (название, адрес, контактные телефоны); ФИО и должность оператора, приявшего заказ; ФИО, должность сотрудника, доставившего заказ.

**Необходимо осуществлять следующую обработку данных:**

- список товаров (код, наименование), пользующихся наибольшим спросом (максимальное количество позиций заказов) у населения за заданный период;

- динамика изменения стоимости заданного товара за заданный период по месяцам;

- список наименований улиц, на которых проживают абоненты предприятия по убыванию числа абонентов.

**3. Автоматизированная информационная система «Прохождение преддипломной практики студентами вуза»**

**Описание предметной области.**

Студенты высшего учебного заведения (Код, Название, Краткое название) в период подготовки дипломной работы (проекта) проходят преддипломную практику. Для каждого студента (Номер зачетной книжки, ФИО), обучающегося на определенной специальности (Код, Название, Краткое название), факультете (Код, Название, Краткое название), форме обучения (Код, Название, Краткое название) фиксируется место прохождения преддипломной практики – предприятие (Код, Название, Краткое название), адрес предприятия, ФИО, должность руководителя от вуза, ФИО, должность руководителя от предприятия, срок прохождения практики (Дата начала, Дата окончания). В базе данных также необходимо вести данные о сроках защиты практики для каждой группы, оценке, полученной студентом за практику. При вводе данных о месте прохождения практики для каждого студента необходимо помечать – планирует ли студент в дальнейшем работать на данном предприятии, варианты ответов - да, нет, не знаю.

**Необходимо осуществлять следующую обработку данных:**

- количество студентов, проходивших практику на заданном предприятии в заданный период;

- перечень предприятий (название, адрес) по алфавиту, на которых проходили преддипломную практику студенты заданной специальности за заданный период;

- на заданную дату список студентов заданной специальности и потока (год обучения), не имеющих оценку за практику.

**4 Автоматизированная информационная система «Лицензионное программное обеспечение организации»**

**Описание предметной области.**

Необходимо вести учет и анализ информации о лицензионном программном обеспечении (ПО), установленном на компьютерах организации (Код, Название, Краткое название). Для каждого компьютера фиксируется инвентарный номер, тип (рабочая станция или сервер), местоположение – в каком подразделении (Код, Название, Краткое название) организации компьютер установлен. Компьютеры могут передаваться из подразделения в подразделение, при этом необходимо знать сроки (Дата начала, Дата окончания) нахождения компьютера в подразделении и на основании какого документа он перемещается (Номер документа, Дата документа), тип этого документа (приказ, распоряжение и т.п.). При установке лицензионного ПО фиксируется, куда установлено ПО – на какой компьютер, название продукта, его тип (среда разработки прикладных программ, среда администрирования БД, операционная система, антивирусная программа и т.п.), фирма производитель, срок действия лицензии (Дата начала, Дата окончания), дата установки, цена за единицу ПО. При этом также необходимо фиксировать информацию об организации, продавшей программное обеспечение – название, адрес, контактные телефоны, адрес сайта.

**Необходимо осуществлять следующую обработку данных:**

- на заданную дату список подразделений, на компьютерах которых установлено не лицензионное ПО;

- список лицензионного ПО, количество лицензий на это ПО (по убыванию) на заданную дату;

- список подразделений, количество компьютеров у подразделения (по убыванию) на заданную дату.

**5 Автоматизированная информационная система «Арендная плата за нежилые помещения»**

**Описание предметной области.**

Организация (Код, Название, Краткое название, Адрес, Контактные телефоны, электронный адрес) сдает в аренду помещения. Каждое помещение характеризуется следующими показателями:

- адрес;

- площадь – кв.м.;

- площадь подвала – кв.м. (при наличии);

- коэффициент подвала – значение от 0 до 1;

- коэффициент технического обустройства помещения (КТ) – значение от 1 до 2.

Арендная плата зависит от базовой ставки за 1 кв.м. (в рублях), которая утверждается документом (Номер, Дата) агентства Госкомимущества России.

Формула расчета месячной арендной платы (МАП):

МАП = (базовая ставка/12 \* площадь помещения + базовая ставка/12 \* площадь подвала \* коэффициент подвала) \* КТ.

При изменении базовой ставки МАП изменяется со следующего месяца после даты изменения ставки. Оплата производится ежемесячно.

Договор об аренде может заключаться как с организациями (Юридическими лицами), так и с физическими лицами. В договоре об аренде помещения, имеющего номер, дату фиксируется дата начала аренды, дата заключения аренды. Для юридического лица в БД заносятся название, адрес, ИНН, номер и дата лицензии о деятельности. Для физического лица – ФИО, паспортные данные (Серия, Номер, Дата выдачи, Кем выдан), ИНН и адрес.

**Необходимо осуществлять следующую обработку данных:**

- итоговая сумма оплат за текущий месяц (на заданную дату);

- список арендаторов (тип, название, адрес и другие характеристики арендуемого помещения) на текущую дату;

- список помещений, не сданных в аренду на текущую дату.

**6 Автоматизированная информационная система «Списание основных средств»**

**Описание предметной области.**

Основные средства - это имущество организации, предприятия со сроком полезного использования. На предприятии (Код, Название, Краткое название) имеется перечень основных средств разного типа (мебель, вычислительная техника, оборудование, инструменты и т.п.), закрепленных за подразделениями предприятия. Закрепление осуществляется на основании определенного документа, имеющего номер, дату, в нем указан срок закрепления средства за подразделением. При списании имущества предприятия создается комиссия, в которую входят руководитель предприятия, главный бухгалтер, главный инженер, главный энергетик, главный механик, руководитель подразделения, где находится средство, материально ответственный в подразделении. При списании средства формируется документ, имеющий номер, название, дату и подписи членов комиссии. В каждом документе может быть указано сразу несколько списываемых средств, для каждого указывается:

- инвентарный номер;

- название;

- принадлежностью к типу;

- дата постановки на учет в подразделении;

- плановый срок эксплуатации (год, месяц);

- балансовая стоимость (в рублях), определяемая при постановке средства на учет.

Для каждого средства также указывается дефект, ставший причиной списания (Код, Название) – износ, поломка, не имеющая восстановления, утрата и др.

**Необходимо осуществлять следующую обработку данных:**

- на заданную дату список (наименование) средств, закрепленных за каждым подразделением, балансовая стоимость средства;

- динамика списания средств заданного наименования (количество) за заданный период по месяцам;

- на заданную дату список комиссии по списанию.

**7. Автоматизированная информационная система «Аттестация сотрудников предприятия»**

**Описание предметной области.**

Предприятие (Код, Название, Краткое название) периодически проводит аттестацию сотрудников на соответствие ими занимаемой должности. Каждый сотрудник за время работы может проходить несколько аттестаций.

Для проведения аттестации (Дата) необходима следующая информация: ФИО сотрудника, дата рождения, место работы (Код, Название, Краткое название) подразделения, занимаемая должность (Код, Название, Краткое название), ставка, дата начала работы, дата окончания работы контракта), название, номер и дата приказа о назначении на должность. Необходимы также следующие сведения:

- сведения об образовании – какое заведение окончил, документ об образовании, квалификация по образованию (инженер, учитель, экономист);

- дата начала трудового стажа;

- дата начала стажа по специальности;

- сведения о повышении квалификации – в каком заведении проходил, дата начала, дата окончания прохождения.

У каждого сотрудника может быть несколько документов об образовании и повышении квалификации.

Каждому аттестуемому могут задать несколько вопросов, необходимо хранить количество заданных вопросов и количество правильных ответов. Также необходимо хранить оценку деятельности работника – соответствует или не соответствует занимаемой должности.

Каждую аттестацию проводит комиссия, необходимо фиксировать ФИО, место работы и должность члена комиссии. Максимальное число – 5 человек.

**Необходимо осуществлять следующую обработку данных:**

- на заданную дату список сотрудников (ФИО, место работы), не прошедших аттестацию – не соответствующих занимаемой должности;

- на заданную дату количество сотрудников, работающих на предприятии в заданной должности;

- список учебных заведений, предприятий, их адреса, на которых сотрудники предприятия повышали свою квалификацию.

**8. Автоматизированная информационная система «Трудоустройство»**

**Описание предметной области.**

Организация (Код, Название, Краткое название Адрес, Контактные телефоны, электронный адрес) предоставляет услуги по трудоустройству. Организацией ведется банк данных о существующих вакансиях. По каждой вакансии поддерживается следующая информация:

- предприятие (Код, Название, Краткое название Адрес, Контактные телефоны, электронный адрес);

- название вакансии (должность);

- требования к соискателю: пол, возраст (Верхняя граница, Нижняя граница), образование (высшее, среднее, не имеет значение и т.п.), знание определенных видов деятельности (выбор из перечня - знание электронного документооборота, определенных прикладных программ и т.п.), коммуникабельность (да, нет);

- обязанности (выбор из перечня – заключение договоров, распространение агитационного материала, работа с клиентами и т.п.);

- предполагаемая оплата (Нижняя граница, Верхняя граница), единицы измерения оплаты - рубли;

- оформление трудовой книжки (да, нет);

- наличие социального пакета (да, нет);

- срок начала открытия вакансии;

- срок закрытия вакансии (вакансия занята).

**Необходимо осуществлять следующую обработку данных:**

- на заданную дату список предприятий, имеющих вакансии по заданной должности;

- название должности, на которую за заданный период было предложено максимальное количество вакансий;

- на заданную дату список предприятий, предлагающих вакансии, не требующих образования.

**9. Автоматизированная информационная система «Спортивные сооружения области»**

**Описание предметной области.**

Областная организация (Код, Название, Краткое название Адрес, Контактные телефоны, электронный адрес) ведет и предоставляет на сайте информацию о спортивных сооружениях области. По каждому сооружению ведется информация:

- место – населенный пункт, городского или сельского типа, адрес;

- номер, название, краткое название;

- тип сооружения (игровые виды спорта, легкоатлетический манеж, каток, ипподром и др.);

- площадь спортивной арены, кв.м.;

- вместимость зрителей, чел., тыс. чел.;

- организация (Код, Название, Краткое название Адрес, Контактные телефоны, электронный адрес), принявшая сооружение на баланс;

- дата принятия на баланс.

Каждое сооружение за время функционирования может находиться на балансе у разных организаций в разные периоды времени.

Необходимо также фиксировать мероприятия, проводимые в спортивных сооружениях:

- тип мероприятия – тренировочный процесс, соревнования, сдача в аренду, концерт и т.п.;

- название мероприятия;

- дата начала, дата окончания мероприятия;

- количество человек, посетивших мероприятие.

**Необходимо осуществлять следующую обработку данных:**

- на заданную дату список спортивных сооружений заданного типа;

- за заданный период динамика занятости спортивного сооружения в мероприятиях заданного типа по месяцам;

- на заданную дату список организаций, на балансе у которых находятся спортивные сооружения, их количество.

**10 Автоматизированная информационная система «Справочник предприятия»**

**Описание предметной области.**

Для формирования контактов организации, имеющей большой контингент клиентов, и представления их на сайте, необходимо хранить следующую информацию:

- код, название краткое название предприятия, каждого его подразделения, взаимодействующего с клиентами;

- вид деятельности предприятия, подразделения – работа с абонентами, изготовление продукции; изучение рынка спроса; IT-подразделение и др.;

- местоположение предприятия, подразделения – адрес, вплоть до номера комнаты. Местоположение может меняться, необходимо отслеживать все данные, для этого фиксируется дата начала закрепления адреса за предприятием, подразделением;

- контактные телефоны – их может быть несколько, и они могут меняться, необходимо хранить историю закрепления телефонов;

- электронный адрес предприятия. Подразделения;

- ФИО, должность руководителя. Руководители также могут меняться, необходимо отслеживать историю их изменения и поддерживать исторические данные.

**Необходимо осуществлять следующую обработку данных:**

- на заданную дату список контактных телефонов подразделений предприятия;

- на заданную дату количество подразделений, не имеющих электронные адреса;

- название подразделения, у которого за заданный период сменилось наибольшее число руководителей.

**11 Автоматизированная информационная система «Паспорт здоровья сотрудника»**

**Описание предметной области.**

Организация придает большое значение здоровью сотрудников и имеет в своей структуре подразделение, занимающееся профилактикой здоровья сотрудников. Для учета состояния здоровья для каждого сотрудника ведется «Паспорт здоровья», в котором сохраняется следующая информация:

- ФИО сотрудника, пол, дата рождения;

- образование (высшее, среднее, без образования). Если человек за время работы на предприятии повышал своё образование – необходимо фиксировать все соответствующие данные, привязывая их к дате получения соответствующего документа;

- история всех перемещений сотрудника на предприятии – подразделение, должность, категория должности (инженерно-технический работник, рабочий, управленческий персонал, IT-специалист и др.), должность, ставка, дата начала работы, дата окончания;

- история семейного положения – состояние (холост, в браке, разведен и др.), дата начала семейной жизни, дата окончания;

- история антропологических измерений – на дату – рост, вес;

- история прививок – дата, название прививки;

- история заболеваний – название, дата постановки на учет, дата снятия с учета.

**Необходимо осуществлять следующую обработку данных:**

- на заданную дату название заболевания, зафиксированного у сотрудников за все время наблюдения максимальное число раз;

- на заданный период список сотрудников, не сделавших прививку заданного вида;

- за заданный период динамика количества заболеваний в организации – по месяцам, количество заболевших с высшим, средним образованием и без образования.

**12 Автоматизированная информационная система «Справочник абитуриента»**

**Описание предметной области.**

Высшее учебное заведения для предоставления на сайте информации абитуриентам ведет банк данных со следующей информацией:

- список специальностей (Код, Название, Краткое название), на которых осуществляется обучение в вузе. Специальности привязаны к учебным подразделениям – факультетам, кафедрам (Код, Название, Краткое название), и распределены по формам обучения (очная, очно-заочная, заочная);

- адрес учебных подразделений;

- телефоны учебных подразделений;

- если есть – адрес сайта учебного подразделения;

- ФИО, ученая степень, ученое звание руководителя учебного заведения (декан факультета, заведующий кафедрой). При этом необходимо вести историю всех руководителей – дата начала работы, дата окончания;

- по каждой форме обучения:

- план приема на специальность на каждый год;

- перечень предметов, по которым необходимо сдавать вступительные экзамены (ЕГЭ);

- проходной балл на специальность по годам с разбивкой по предметам.

**Необходимо осуществлять следующую обработку данных:**

- на заданный год – список специальностей заданной формы обучения и планы приема;

- на заданный год наименование специальности, на которую был максимальный проходной балл по математике;

- на заданный год список руководителей учебных подразделений, имеющих ученую степень «доктор наук» и ученое звание «профессор».

**13 Автоматизированная информационная система «Платные образовательные услуги населению»**

Описание предметной области.

Организация (Код, Название, Краткое название) оказывает платные образовательные услуги населению. Услуги оказываются в виде проведения курсов обучения, по которым необходимо хранить следующую информацию:

- тип проведения – групповые, индивидуальные;

- вид проведения – очные, заочные;

- дата начала, дата окончания курсов;

- срок обучения (дни, месяцы, годы);

- количество часов обучения;

- на базе какого образования (среднее, высшее);

- темы, входящие в курс, для каждой темы:

название;

количество часов;

- время проведения занятий – дни недели, часы;

- вид выпускного контроля (квалификационная работа, экзамен, собеседование и прочее);

- вид выдаваемого документа (документ государственного образца, документ установленного образца);

- стоимость обучения

Для предоставления информации на сайте необходимо хранить адрес организации, контактные телефоны, электронный адрес, адрес сайта, серия, номер и вид документа о предоставлении образовательных услуг.

**Необходимо осуществлять следующую обработку данных:**

- список курсов, на которых можно прослушать заданную темы, например, «1С Бухгалтерия»;

- список курсов, на которых можно пройти заочное обучение и имеющих минимальную стоимость;

- список самых длительных курсов.

**14 Автоматизированная информационная система «Новостная лента организации»**

**Описание предметной области.**

Для предоставления новостных событий организации на её сайте необходимо вести следующие данные:

- название, краткое название организации, контактные телефоны, адрес, электронный адрес, адрес сайта;

- название и координаты подразделений организации, информация о которых будет предоставляться на сайте;

- список работающих сотрудников подразделений организации. Которым предоставляется возможность размещать информацию на сайте – ФИО, подразделение, должность, логин, пароль. При изменении статуса сотрудника – увольнение, перевод – информация должна соответствующим образом изменяться, например, сотрудник переводится в статус неработающего, логин и пароль д.б. заблокированы;

- описание новостной информации, размещаемой на сайте:

тип (новость, объявление, сообщение и др.);

название информации;

дата создания;

текст;

дата размещения;

дата перевода информации в архив;

размер информации в Кб;

наличие прикрепляемых к информации файлов – для каждого название, размер, тип, краткое описание;

ответственный за информацию – сотрудник подразделения, имеющий соответствующий доступ.

**Необходимо осуществлять следующую обработку данных:**

- на заданную дату список ответственных за информацию на сайте от подразделений, не имеющих логин и пароль;

- на заданную дату название информации, размещенной на сайте (не в архиве) и имеющей самый большой размер.

- динамика предоставления информации для сайта заданным подразделением за заданный период – количество по месяцам.

**15 Автоматизированная информационная система «Анализ продаж»**

**Описание предметной области.**

Магазин (Код, Название, Краткое название) ведет учет продаж товаров и анализ работы с постоянными клиентами. Каждая единица товара учитывается при поступлении в магазин из накладной (Номер, Дата накладной), которая может иметь несколько позиций. В каждой позиции есть её номер, наименование товара, количество единиц поступившего товара, единица измерения, цена за единицу. Товары учитываются по виду - одежда, кожгалантерея, чулочно-носочные изделия, обувь и т.п. Каждый товар также имеет определенный артикул.

Ведет учет и продаж товаров – фиксируется дата продажи конкретного товара, количество проданных единиц.

Магазин ведет учет постоянных клиентов – фиксируется ФИО клиента, его паспортные данные (Серия, Номер, Дата выдачи, Кем выдан), дата рождения, контактный телефон. Покупателю, сделавшему покупку на сумму свыше 3000 тыс. рублей выдается дисконтная карта, имеющая 5-ти значный номер. Карта дает покупателю скидку 3%. При накоплении сумм покупок покупателем более чем на 10000 тыс. рублей, процент скидки увеличивается до 5%, более 20000 – максимальный процент скидки достигает размера 10%.

**Необходимо осуществлять следующую обработку данных:**

- на заданную дату количество и список покупателей (ФИО, контактный телефон), имеющих 10% скидку;

- за заданный период - динамика продажи заданного товара – количество по месяцам – поступление/ продажа;

- на заданную дату список покупателей (ФИО, контактный телефон), у которых в ближайшие 10 дней будет день рождения.

**16 Автоматизированная информационная система «Электронный реестр помещений»**

**Описание предметной области.**

Предприятие (Код, Название, Краткое название) имеет иерархическую организационную структуру, отражающая подчиненность большого количества подразделений. Для каждого подразделения необходимо хранить:

- код, полное название, краткое название;

- родительные и дательные падежи названий для автоматизированного формирования ряда документов и отчетов.

Каждое подразделение может занимать несколько помещений. Каждое помещение имеет номер, в который входит номер корпуса (предприятие может иметь много зданий – 1 или 2 цифры) и номер этажа – 1 или 2 цифры. В пределах одного этажа каждое помещение имеет свой номер 1 или 3 цифры. Помещение относится к определенному типу, о котором также необходимо иметь сведения, например, кабинет руководителя, приемная руководителя, лаборатория, цех, столовая и т.п. Необходимо также хранить данные о площади каждого помещения (кв. м).

Закрепление помещений за подразделениями может изменяться. Это осуществляется на основе определенного документа, имеющего название (приказ, распоряжение) и дату. В каждом документе м.б. несколько позиций, отображающих следующую информацию: номер позиции документа; действие, осуществляемое с помещением (передать, закрепить) дата действия; название подразделения; перечень помещений, возможное наименование другого подразделения. Например – «передать с 20.06.2007 г. отделу № 3 лабораторные помещения 14105 и 14106, закрепленные за лабораторией № 5»; «закрепить за медпунктом с 15.09.2007 г. складской помещение 3109» .

**Необходимо осуществлять следующую обработку данных:**

- на заданную дату список подразделений предприятия (наименование) и перечень занимаемых им помещений – номер, тип;

- список, отображающий иерархию (дерево) подчинения подразделений предприятия;

- динамика изменения количества площадей помещений у заданного подразделения за заданный период – количество по годам.

**17 Автоматизированная информационная система «Скорая помощь»**

**Описание предметной области.**

Лечебное учреждение (Код, Название, Краткое название, Адрес, Контактные телефоны) оказывает скорую медицинскую помощь населению. В учреждении имеется штат сотрудников, о которых необходимо хранит следующие сведения:

- табельный номер;

- ФИО; дата рождения, пол;

- должность, дата начала работы в данной должности, дата окончания, ставка.

Работа в учреждении круглосуточная – сотрудники работают по 24 часа с последующими выходными днями. Необходимо знать, в какой смене и бригаде работает тот или иной сотрудник. Закрепление в бригаду осуществляется на основании внутреннего приказа, имеющего номер и дату. В каждой позиции приказа указывается, что конкретный сотрудник с даты 1 по дату 2 работает в бригаде с заданным номером.

Необходимо вести учет сведений о выездах бригад на вызовы. Каждый вызов определяется датой, временем выезда и адресом. Пациент, которому оказывается помощь, может быть описан следующими данными ФИО, возраст (примерный), первоначальный диагноз. Необходимо также знать ФИО и должности сотрудников выехавшей на вызов бригады (включая водителя и диспетчера). Необходимо также хранить небольшое текстовое описание принятых бригадой мер. Если больной был госпитализирован, либо получил направление на госпитализацию, также необходимо знать в какое учреждение он был направлен (название, адрес). При возвращении бригады фиксируется время прибытия.

**Необходимо осуществлять следующую обработку данных:**

- на заданную дату список выездов всех бригад учреждения (номер выезда, время, номер бригады, принятые меры);

- на заданную дату описание самого длительного выезда;

- на заданную дату список заданной бригады (табельный номер, ФИО, должность).

**Список использованных источников**

1. Буч, Г. Язык UML. Руководство пользователя. 2-е изд. / Г. Буч, Д. Рамбо, И. Якобсон ; пер. с англ. Н. Мухин -- М.: ДМК Пресс, 2013. -- 496с.: ил.
2. Вендров А.М. Case-технологии. Современные методы и средства проектирования информационных систем. - М. Финансы и статистика, 2012 - 176 с.: ил.
3. Леоненков А. Самоучитель UML- СПб.:BHV, 2011.
4. Арлоу Д., Нейштадт И. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование, 2-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2012. – 624 с., ил.
5. Вендров А.М. Практикум по проектированию программного обеспечения экономических информационных систем: Учеб. Пособие. - М.: Финансы и статистика, 2014. - 192 с.: ил.
6. Леоненктов А.В. Обектно-ориентированный анализ и проектирование с использованием UMLRationalRose: Учебное пособие / А.В. Леоненков. – И.: Интернет-Университет Информационных Технологий; БИНОМ. Лабораториязнаний, 2016. – 320 с.: ил. – (Серия «Основыинформационныхтехнологий)
7. ФаулерM.UML. Основы, 33е издание. – Пер. с англ. – СПб: Символ3Плюс, 2014. – 192 с.
8. Рамбо, Дж. UML 2.0. Объектно-ориентированное моделирование и разработка / Дж. Рамбо. СПб.: Питер, 2016. 544 с.
9. Дж. Рамбо, М. Блоха. UML 2.0. Объектно-ориентированное моделирование и разработка. 2-е изд. – СПб.: Питер, 2012. – 544 с.: ил.
10. [Визуальное моделирование: теория и практика: Информация] http://www.intuit.ru/studies/courses/1041/218/info