**ЛАБОРАТОРНАЯ РАБОТА № 10-11**

**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.

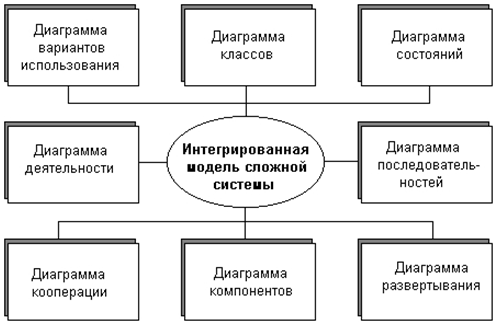
*UML* представляет собой *объектно-ориентированный* язык моделирования, обладающий следующими основными характеристиками:

* 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.



#### *Рис. 1. Интегрированная модель сложной системы в нотации языка UML*

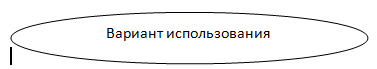
Стандарт UML предлагает  следующий  набор  диаграмм для моделирования:

* 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:



### *Рис.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

*Рис.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).

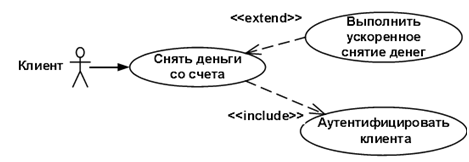
Связь  коммуникации –  это  связь  между  вариантом  использования и действующим лицом. На языке UML связи коммуникации показывают с помощью однонаправленной ассоциации (сплошной линии).



### *Рис.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.



### *Рис.5. Пример связи включения и расширения*

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

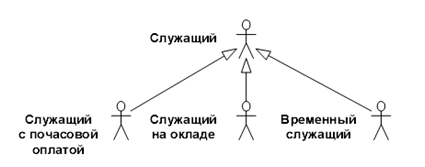


Рис.6. Пример связи обобщения

**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.

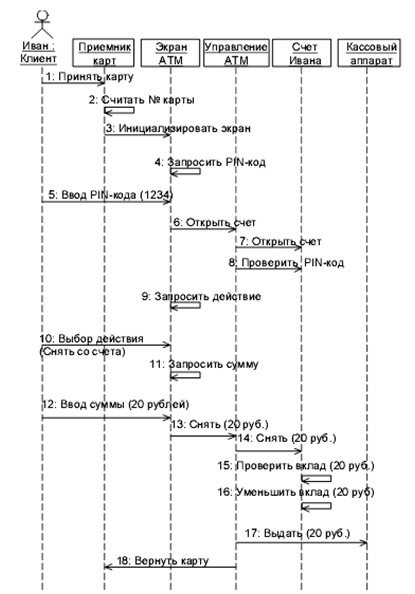


Рис. 7. Пример диаграммы последовательности

**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.

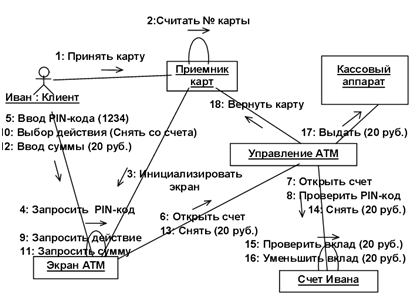


Рис. 8. Пример диаграммы кооперации

**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).

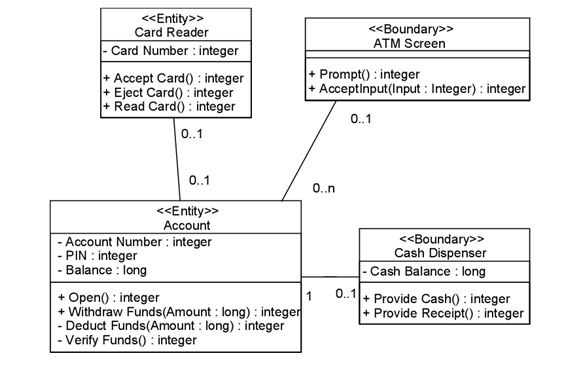


Рис. 9. Пример диаграммы классов

**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.



Рис. 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.

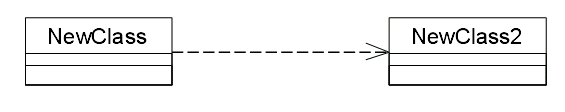


Рис. 11. Связь зависимость

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:



Рис. 11. 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.

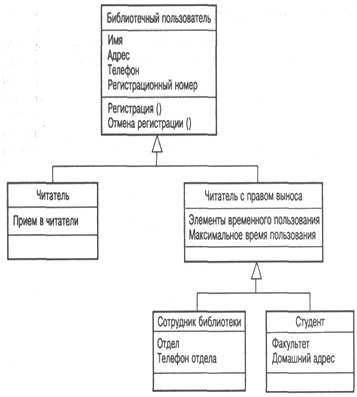


Рис. 12. Пример связи наследование

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):



Рис. 13. 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:

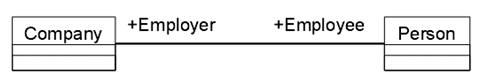


Рис. 14. Пример ролей  связей

**Пакет. Механизм пакетов**

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

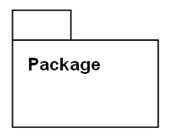


Рис. 15. Обозначение пакета в UML

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.

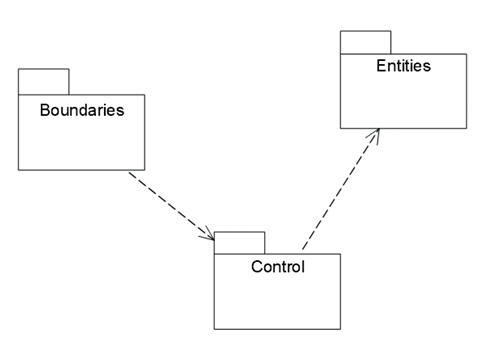


Рис. 16. Пример диаграммы пакетов

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 « ^ ».

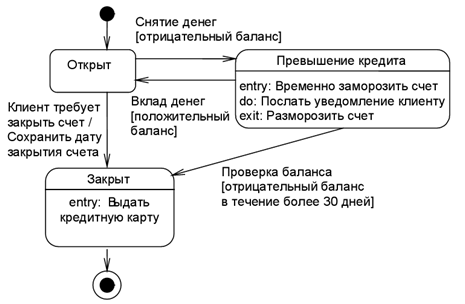


Рис. 17. Пример диаграммы состояний

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.

**Диаграммы размещения**

Диаграмма размещения (deploymentdiagram) отражает физические взаимосвязи  между  программными  и  аппаратными  компонентами системы.  Она  является  хорошим  средством  для  того,  чтобы  показать маршруты  перемещения  объектов  и  компонентов  в  распределенной системе.

Каждый  узел  на  диаграмме  размещения  представляет  собой некоторый  тип  вычислительного  устройства –  в  большинстве  случаев, часть аппаратуры. Эта аппаратура может быть простым устройством или датчиком, а может быть и мэйнфреймом.

Диаграмма размещения показывает физическое расположение сети и местонахождение  в  ней  различных  компонентов.

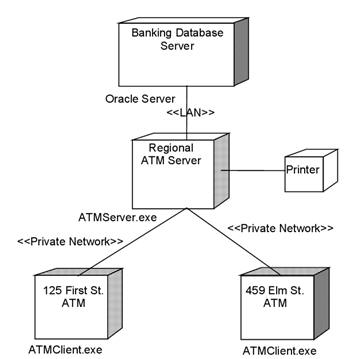


Рис. 19. Пример диаграммы размещения

Диаграмма  размещения  используется  менеджером  проекта, пользователями, архитектором системы и эксплуатационным персоналом, чтобы  понять  физическое  размещение  системы  и  расположение её отдельных подсистем.

**Диаграммы компонентов**

Диаграммы  компонентов  показывают,  как  выглядит  модель на физическом  уровне.  На  них  изображены  компоненты  программного обеспечения и связи между ними. При этом на такой диаграмме выделяют два типа компонентов: исполняемые компоненты и библиотеки кода.

Каждый класс модели (или подсистема) преобразуется в компонент исходного  кода.  После  создания  они  сразу  добавляются  к  диаграмме компонентов. Между отдельными компонентами изображают зависимости, соответствующие  зависимостям  на  этапе  компиляции  или  выполнения программы.

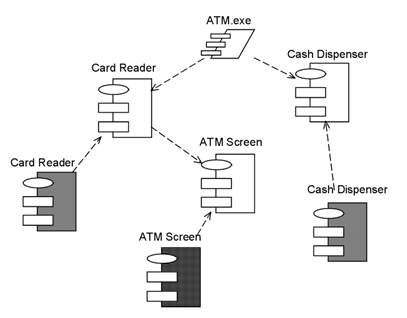


Рис. 18. Пример диаграммы компонентов

Диаграммы компонентов применяются теми участниками проекта, кто отвечает за компиляцию системы. Из нее видно, в каком порядке надо компилировать  компоненты,  а  также  какие  исполняемые  компоненты будут  созданы  системой.  На  такой  диаграмме  показано  соответствие классов  реализованным  компонентам.  Она  нужна  там,  где  начинается генерация кода.

**Объединение диаграмм  компонентов и развертывания**

В некоторых случаях допускается размещать диаграмму компонентов на диаграмме развертывания. Это позволяет показать какие компоненты выполняются и  на каких узлах.

# Практическая часть

1. Изучить предлагаемый теоретический материал.
2. Постройте диаграмму вариантов использования для выбранной информационной системы **(Варианты заданий на стр. 41-45).**
3. 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 № 12-13

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

Цель работы: Освоить методику построения диаграмм классов;

### **Теоретическая часть**

Class diagram (диаграмма классов) — основная диаграмма для создания кода приложения. При помощи диаграммы классов создается внутренняя структура системы, описывается наследование и взаимное положение классов друг относительно друга. Здесь описывается логическое представление систе­мы. Именно логическое, так как классы — это лишь заготовки, на основе ко­торых затем будут определены физические объекты.

Таким образом, диаграмма классов описывает общее представление систе­мы и является противоположной Collaboration diagram, в которой представле­ны объекты системы. Однако такое разделение не является строгим правилом, и возможно смешанное представление классов и объектов.

**Особенности разработки диаграмм классов в среде IBM Rational Rose 2003**

Диаграмма *классов* является основным логическим представлением модели и содержит детальную информацию о внутреннем устройстве объектно-ориентированной программной системы или, используя современную терминологию, об архитектуре программной системы. Активизировать рабочее окно диаграммы *классов* можно несколькими способами:

- окно диаграммы *классов* появляется по умолчанию в рабочем окне диаграммы после создания нового проекта;

- щелкнуть на кнопке с изображением диаграммы классов на стандартной панели инструментов;

- раскрыть логическое представление (Logical View) в браузере проекта и дважды щелкнуть на пиктограмме Main (Главная);

- выполнить операцию главного меню: BrowseОписание: srarrClass Diagram (ОбзорОписание: srarrДиаграмма *классов*).

При этом появляется новое окно с чистым рабочим листом диаграммы *классов* и специальная панель инструментов, содержащая кнопки с изображением графических примитивов, необходимых для разработки диаграммы *классов* (табл. 1). Назначение отдельных кнопок панели можно узнать также из всплывающих подсказок.

Таблица 4.1 - Назначение кнопок специальной панели инструментов для диаграммы классов

|  |  |  |
| --- | --- | --- |
| **Графическое изображение** | **Всплывающая подсказка** | **Назначение кнопки** |
| Описание: 4_1 | Selection Tool | Превращает изображение курсора в форму стрелки для последующего выделения элементов на диаграмме |
| Описание: 4_2 | Text Box | Добавляет на диаграмму текстовую область |
| Описание: 4_3 | Note | Добавляет на диаграмму примечание |
| Описание: 4_4 | Anchor Note to Item | Добавляет на диаграмму связь примечания с соответствующим графическим элементом диаграммы |
| Описание: 4_5 | Class | Добавляет на диаграмму *класс* |
| Описание: 4_6 | Interfase | Добавляет на диаграмму *интерфейс* |
| Описание: 4_7 | Unidirectional Association | Добавляет на диаграмму направленную *ассоциацию* |
| Описание: 4_8 | Association Class | Добавляет на диаграмму *ассоциацию класс* |
| Описание: 4_9 | Package | Добавляет на диаграмму пакет |
| Описание: 4_10 | Dependency or Instantiates | Добавляет на диаграмму отношение зависимости |
| Описание: 4_11 | Generalization | Добавляет на диаграмму отношение обобщения |
| Описание: 4_12 | Realize | Добавляет на диаграмму отношение реализации |

2.1.1 Добавление класса на диаграмму классов и редактирование его свойств

Для добавления класса на диаграмму классов нужно с помощью левой кнопки мыши нажать кнопку с изображением пиктограммы класса на специальной панели инструментов, отпустить левую кнопку мыши и щелкнуть левой кнопкой мыши на свободном месте рабочего листа диаграммы. На диаграмме появится изображение класса с маркерами изменения его геометрических размеров и предложенным средой именем по умолчанию NewClass.

Продолжая разработку модели приложения для АИС своего варианта, например, «Трудоустройство», в качестве сквозного примера проекта, построим для этой модели следующую диаграмму классов. С этой целью следует изменить предложенное по умолчанию имя диаграммы Main на - Диаграмма классов «Трудоустройство», а имя добавленного на диаграмму класса - на «Matrix» (см. рис. 4.1).

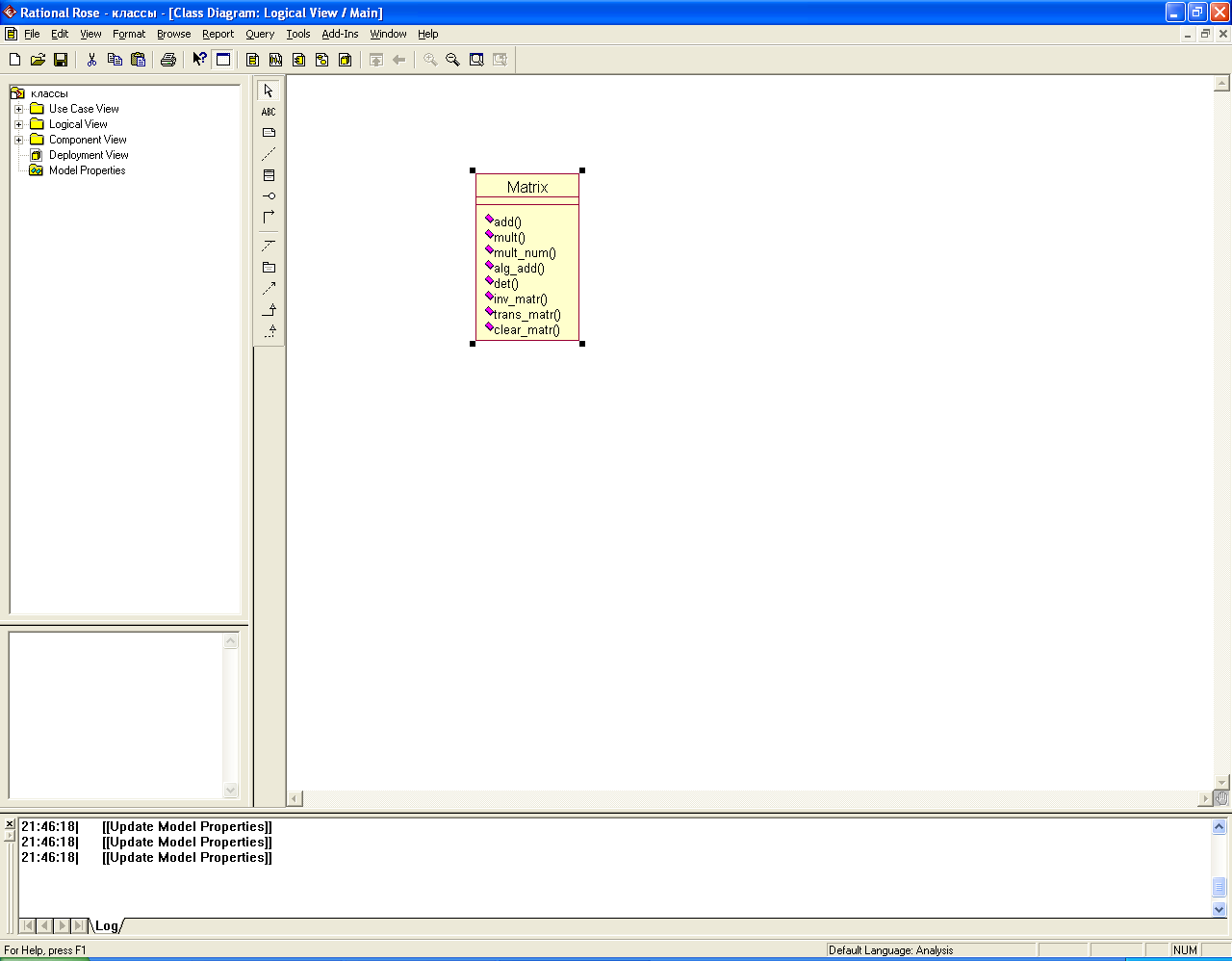


Рисунок 4.1 - Диаграмма классов после добавления на нее класса «Matrix»

Для класса «Matrix» можно уточнить его назначение в модели с помощью указания стереотипа и пояснительного текста в форме документации. С этой целью двойным щелчком левой кнопкой мыши на изображении этого класса на диаграмме или в браузере проекта следует открыть диалоговое окно спецификации свойств этого класса (рис. 4.2) и на вкладке General (Общие) выбрать из вложенного списка Stereotype стереотип.

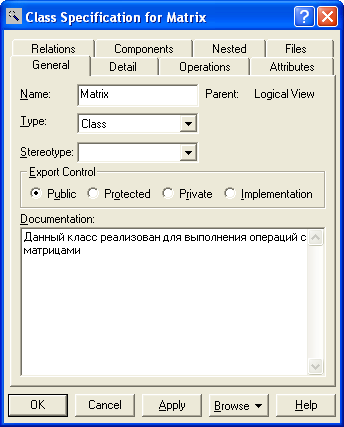


Рисунок 4.2 - Диалоговое окно спецификации свойств класса

Для отдельного класса можно уточнить также и другие его свойства, доступные для редактирования на вкладке Detail (Подробно) окна спецификации свойств этого класса. Например, на этой вкладке с помощью вложенного списка Multiplicity (Кратность) можно задать количество объектов или экземпляров данного класса, для чего следует выбрать строку с буквой n. Данное значение означает, что у класса может быть любое конечное число экземпляров (рис. 4.3). Поле ввода с именем Space (Пространство) служит для указания объема абсолютной или относительной памяти, которая требуется, по оценке разработчика, для реализации каждого объекта данного класса. Применительно к рассматриваемой модели это поле можно оставить пустым.

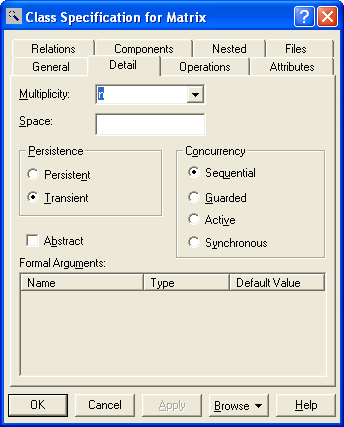


Рисунок 4.3 - Диалоговое окно спецификации свойств класса

Далее можно задать устойчивость классов в группе выбора Persistence. При этом выбор свойства Persistent (Устойчивый) означает, что информация об объектах данного класса должна быть сохранена в системе. Выбор свойства Transient (Временный) означает, что нет необходимости сохранять информацию об объектах данного класса в системе после завершения работы программного приложения. Применительно к рассматриваемой модели следует выбрать свойство Persistent.

В группе выбора Concurrency (Параллельность) можно специфицировать условия на возможность реализации объектов данного класса в параллельных потоках управления. Для выбора могут быть использованы следующие свойства:

Sequential (Последовательный) - свойство по умолчанию, которое означает, что объекты класса будут вести себя нормально только при наличии одного потока управления, т. е. соответствующие операции объектов должны выполняться последовательно. В то же время при наличии нескольких потоков управления стабильное поведение объектов класса не гарантируется.

Guarded (Безопасный) - означает, что при наличии нескольких потоков управления объекты класса будут вести себя ожидаемым от них образом. Для этого объекты в различных потоках должны взаимодействовать друг с другом для того, чтобы гарантировать отсутствие конфликта между ними.

Active (Активный) - означает, что класс должен иметь свой собственный поток управления.

Synchronous (Синхронный) - означает, что объекты класса будут вести себя ожидаемым от них образом при наличии нескольких потоков управления. При этом нет необходимости во взаимодействии объектов в различных потоках управления, поскольку объекты данного класса могут самостоятельно разрешать возможные конфликты.

Для того, чтобы специфицировать класс как абстрактный, т.е. не имеющий экземпляров, следует на этой же вкладке выставить отметку в свойстве Abstract (Абстрактный).

2.1.2 Добавление и редактирование атрибутов классов

Из всех графических элементов среды IBM Rational Rose 2003 класс обладает максимальным набором свойств, главными из которых являются его *атрибуты* и *операции*.

Добавить *атрибут* к созданному ранее классу можно одним из следующих способов:

- с помощью *операции* контекстного меню New Attribute (Новый *атрибут*) для класса, выделенного на диаграмме классов. В этом случае активизируется курсор ввода текста в области графического изображения класса на диаграмме.

- с помощью *операции* контекстного меню: NewОписание: srarrAttribute (НовыйОписание: srarr*Атрибут*) для класса, выделенного в браузере проекта. В этом случае активизируется курсор ввода текста в области иерархического представления класса в браузере проекта под именем соответствующего класса.

- с помощью *операции* контекстного меню Insert (Вставить), вызванного при позиционировании курсора в области открытой вкладки *атрибутов* в диалоговом окне свойств Class Specification соответствующего класса.

После добавления *атрибута* к классу по умолчанию ему присваивается имя name и некоторый *квантор видимости* (рис. 4.4).

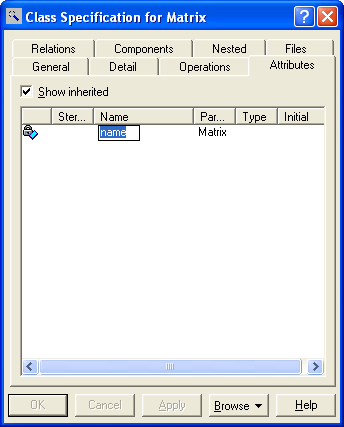


Рисунок 4.4 - Диалоговое окно спецификации свойств

класса после добавления нового атрибута

Напомним, что имена *атрибутов* и *операций* классов должны начинаться со строчной буквы. Видимость *атрибутов* на диаграмме классов изображается в форме специальных пиктограмм. Используемые пиктограммы видимости изображаются перед именем соответствующего *атрибута* и имеют следующий смысл (табл. 4.2).

Таблица 4.2 - Пиктограммы видимости атрибутов классов

|  |  |  |
| --- | --- | --- |
| **Графическое изображение** | **Текстовый аналог** | **Назначение пиктограммы** |
| Описание: 5_2 | Public | Общедоступный или открытый. В нотации языка UML такому атрибуту соответствует знак «+» |
| Описание: 5_3 | Protected | Защищенный. В нотации языка UML такому атрибуту соответствует знак «#» |
| Описание: 5_4 | Private | Закрытый. В нотации языка UML такому атрибуту соответствует знак «-» |
| Описание: 5_5 | Implementation | Реализация. В нотации языка UML такому атрибуту соответствует знак «∼» |

Для редактирования свойств *атрибутов* предназначено специальное диалоговое окно спецификации *атрибута* Class Attribute Specification, которое открывается двойным щелчком мыши на строке выбранного *атрибута* в окне спецификации свойств класса. В окне свойств отдельного *атрибута* класса можно задать *тип* данных *атрибута* и его начальное *значение*, а также назначить *атрибуту* стереотип из раскрывающегося списка или изменить его *квантор видимости*.

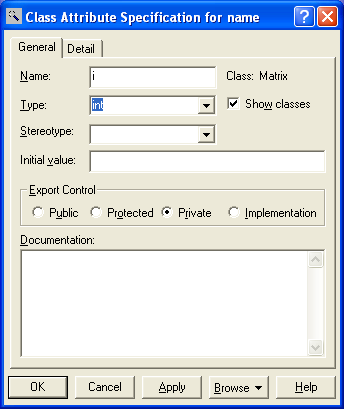


Рисунок 4.5 - Диалоговое окно спецификации свойств атрибута

2.1.4 Добавление и редактирование операций классов

Функционирование любой системы основано на выполнении отдельными его элементами тех или иных действий. В нашей модели все действия представляются с помощью *операций* классов. Таким образом, следующий этап разработки диаграммы классов связан со спецификацией *операций* классов.

Добавить *операцию* к созданному ранее классу можно одним из следующих способов:

- с помощью *операции* контекстного меню New Operation (Новая *операция*) для класса, выделенного на диаграмме классов. В этом случае активизируется курсор ввода в области графического изображения класса на диаграмме.

- с помощью *операции* контекстного меню: NewОписание: srarrOperation (НоваяОписание: srarr*Операция*) для класса, выделенного в браузере проекта. В этом случае активизируется курсор ввода в области иерархического представления класса в браузере под именем соответствующего класса.

- с помощью *операции* контекстного меню Insert (Вставить), вызванного при позиционировании курсора в области открытой вкладки *операций* в диалоговом окне свойств Class Specification соответствующего класса.

После добавления *операции* к классу по умолчанию ей присваивается имя opname и некоторый *квантор видимости*. Видимость *операций* на диаграмме классов также изображается в форме специальных пиктограмм. Используемые пиктограммы видимости изображаются перед именем соответствующей *операции* и имеют следующий смысл (табл. 4.3).

Таблица 4.3 - Пиктограммы видимости операций классов

|  |  |  |
| --- | --- | --- |
| **Графическое изображение** | **Текстовый аналог** | **Назначение пиктограммы** |
| Описание: 5_8 | Public | Общедоступный или открытый. В нотации языка UML такому атрибуту соответствует знак «+» |
| Описание: 5_9 | Protected | Защищенный. В нотации языка UML такому атрибуту соответствует знак «#» |
| Описание: 5_10 | Private | Закрытый. В нотации языка UML такому атрибуту соответствует знак «-» |
| Описание: 5_11 | Implementation | Реализация. В нотации языка UML такому атрибуту соответствует знак «∼» |

Каждая из *операций* классов имеет собственное диалоговое окно спецификации свойств Operation Specification, которое может быть открыто по двойному щелчку на имени *операции* на соответствующей вкладке спецификации класса или на имени этой *операции* в браузере проекта (рис. 4.9).

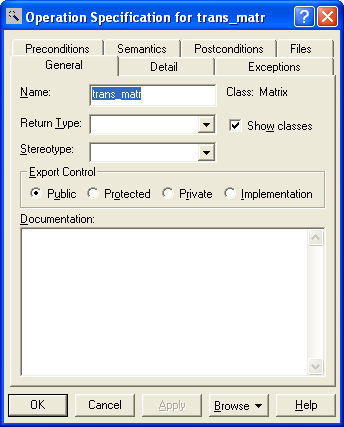


Рисунок 4.6 - Диалоговое окно спецификации свойств операции

Для *операций* классов кроме *квантора видимости* можно также задать: *аргументы* и их тип, *тип возвращаемого результата*, стереотип *операции*, а также определить протокол и размер, задать исключительные ситуации, специфицировать предусловия и постусловия и целый ряд других свойств. Для отдельной *операции* эти дополнительные свойства доступны для редактирования на вкладке Detail (Подробно) диалогового окна спецификации свойств выбранной *операции* (рис. 4.7).

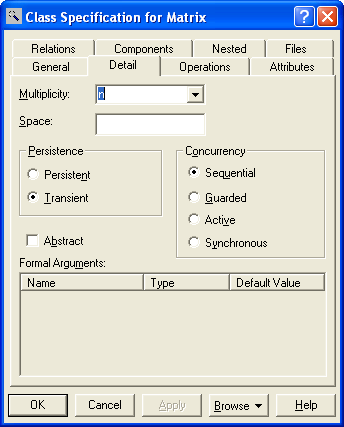


Рисунок 4.7 -   Диалоговое окно спецификации свойств операции, открытое на вкладке Detail (Подробно)

На вкладке Detail в многостраничном поле Arguments (Аргументы) можно определить *аргументы редактируемой операции*. Для этого следует выполнить *операцию* контекстного меню Insert (Вставить). После этого в этом поле появится аргумент данной *операции* с именем по умолчанию argname. Для редактирования свойств аргумента предназначено специальное окно свойств аргумента.

На вкладке Detail в поле Protocol (Протокол) можно специфицировать порядок выполнения *операций* класса, например, указать, что одна *операция* не может быть вызвана раньше другой. Соответствующий текст в данное поле вводится с клавиатуры и попадает в генерируемый код в форме комментария. В поле Qualification (Квалификация) можно уточнить детали реализации *операции*, связанные с конкретным языком программирования. Соответствующий текст также вводится в данное поле с клавиатуры и попадает в генерируемый код в форме комментария.

Далее на этой же вкладке в полях Size (Размер) и Time (Время) можно специфицировать предполагаемый объем памяти и время, необходимое для выполнения *операции*. Соответствующая информация попадает в генерируемый код в форме комментария.

В группе выбора Concurrency (Параллельность) можно специфицировать условия на возможность параллельного выполнения данной *операции*. Для выбора могут быть использованы следующие свойства:

Sequential (Последовательная) - свойство по умолчанию, которое означает, что данная *операция* класса может быть выполнена только при наличии одного потока управления, т. е. соответствующая *операция* класса должна выполняться последовательно. При наличии нескольких потоков управления выполнение данной *операции* класса не гарантируется.

Guarded (Безопасная) - означает, что при наличии нескольких потоков управления выполнение данной *операции* класса гарантируется только в том случае, когда обеспечено взаимодействие объектов друг с другом в различных потоках.

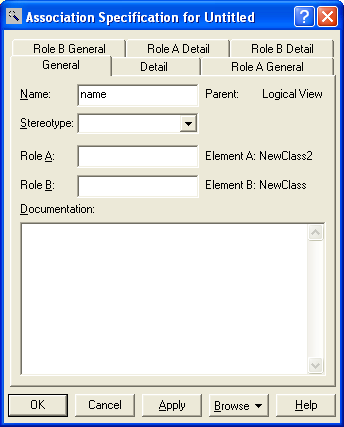
Synchronous (Синхронная) - означает, что выполнение данной *операции* класса гарантируется при наличии нескольких потоков управления. При этом нет необходимости во взаимодействии объектов в различных потоках управления, поскольку данная *операция* класса будет выполняться в отдельном потоке управления вплоть до своего завершения.

2.1.5 Добавление ассоциации на диаграмму классов и редактирование ее свойств

Добавление на диаграмму *ассоциации* между двумя классами выполняется следующим образом. На специальной панели инструментов необходимо нажать кнопку с изображением пиктограммы направленной *ассоциации* и отпустить левую кнопку мыши. Если *ассоциация* - направленная, то на диаграмме классов надо выделить первый элемент *ассоциации* или источник, от которого исходит стрелка, и, не отпуская нажатую левую кнопку мыши, переместить ее указатель ко второму элементу отношения или приемнику, к которому направлена стрелка. После перемещения ко второму элементу кнопку мыши следует отпустить, в результате чего на диаграмму классов будет добавлена направленная *ассоциация* с именем Untitled между двумя выбранными классами (рис. 4.11).

  
Рисунок 4.11 -  Фрагмент диаграммы классов после добавления на неё направленной ассоциации

Изменим имя для данной *ассоциации*, предложенное средой по умолчанию. Это можно выполнить с помощью окна спецификации свойств *ассоциации*. Доступ к диалоговому окну спецификации свойств *ассоциации* Association Specification можно получить после выделения линии *ассоциации* на диаграмме классов или в браузере проекта и двойного щелчка на ней левой кнопки мыши (рис. 4.12).

  
Рисунок 4.12 - Диалоговое окно спецификации свойств ассоциации

Для задания имени *ассоциации* следует на вкладке General (Общие) в поле ввода Name (Имя) ввести текст ее имени: Соответствует и нажать кнопку Apply или OK, чтобы сохранить результаты редактирования имени *ассоциации*. Для *ассоциации* можно задать также *кратность* каждого из концов *ассоциации*, стереотип, использовать ограничения и роли, а также некоторые другие свойства.

Если *ассоциация* является ненаправленной, то порядок выбора классов может быть произвольный, а после добавления *ассоциации* на диаграмму классов следует изменить значение соответствующего свойства данной *ассоциации*. С этой целью необходимо перейти на вкладку Role A Detail в окне спецификации свойств *ассоциации* и убрать отметку у свойства Navigable (Навигация).

2.1.6 Добавление отношений агрегации и композиции на диаграмму классов и редактирование их свойств

Агрегация – это частный случай ассоциации, описывающий объекты, состоящие из частей. Например, газонокосилка состоит из ножа, двигателя, нескольких колес и корпуса. Здесь газонокосилка является агрегатом (незакрашенный ромбик), а остальные детали – составляющими частями (стрелка).

Композиция – это частный случай агрегации. Композиция подразумевает, что части принадлежат целому. Например, компания, состоящая из отделений, которые в свою очередь, состоят из отделов.

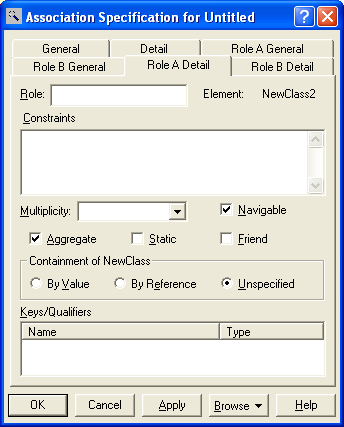
- добавить на диаграмму отношение *агрегации* между двумя классами можно следующими способами:

- щелкнуть на кнопке с изображением отношение *агрегации* на специальной панели инструментов и провести линию *агрегации* от одного класса к другому.

Провести линию *ассоциации* между выбранными классами и изменить ее свойства таким образом, чтобы превратить данную *ассоциацию* в *агрегацию*.

В первом случае может оказаться, что по умолчанию на специальной панели инструментов диаграммы классов отсутствует кнопка с пиктограммой *агрегации*. В этом случае необходимо предварительно добавить ее на панель инструментов одним из описанных ранее способов. Во втором случае следует открыть окно спецификации свойств *ассоциации* Association Specification и на вкладке деталей соответствующего конца *ассоциации* выставить отметку в строке выбора Aggregate (*Агрегация*).

В качестве примера изменим тип созданной ранее *ассоциации* и сделаем ее агрегацией. С этой целью на вкладке Role В Detail деталей конца *ассоциации* одного класса следует выставить отметку в строке выбора Aggregate(рис. 4.13).

  
Рисунок 4.13 - Диалоговое окно спецификации свойств ассоциации

Соответствующий фрагмент диаграммы классов после изменения *ассоциации* между классами на отношение *агрегации* будет иметь следующий вид (рис. 4.14).

  
Рисунок 4.14 -  Фрагмент диаграммы классов модели после добавления на нее отношения агрегации

Для изображения отношения *композиции* можно также вначале изобразить обычную *ассоциацию*, после чего, открыв окно ее свойств на вкладке деталей соответствующего конца *ассоциации*, выставить отметку в строке выбора Aggregate (*Агрегация*) и в секции Containment (Локализация) выбрать опцию By Value (По значению). По умолчанию эта опция не специфицирована, т.е. выставлена отметка опции Unspecified (рис. 4.15).



Рисунок 4.15 -  Фрагмент диаграммы классов модели после добавления на нее отношения композиции

2.1.7 Добавление отношения обобщения на диаграмму классов и редактирование ее свойств

Добавление на диаграмму отношения *обобщения* между двумя классами выполняется следующим образом. На специальной панели инструментов необходимо нажать кнопку с изображением пиктограммы *обобщения* и отпустить левую кнопку мыши. Далее на диаграмме классов надо выделить первый элемент *обобщения* или потомок, от которого исходит стрелка, и, не отпуская нажатую левую кнопку мыши, переместить ее указатель ко второму элементу отношения или предок, к которому направлена стрелка. После перемещения ко второму элементу кнопку мыши следует отпустить, в результате чего на диаграмму классов будет добавлена линия *обобщения* с именем Untitled между двумя выбранными классами. Пример приведен на рисунке 4.16.

  
Рисунок 4.16 – Пример диаграммы классов после добавления на неё

отношения обобщения

Изменим имя отношения *обобщения*, предложенное средой по умолчанию. Это можно выполнить с помощью окна спецификации свойств *обобщения*. Доступ к диалоговому окну спецификации свойств отношения *обобщения* Generalize Specification можно получить после выделения линии *обобщения* на диаграмме классов или в браузере проекта и двойного щелчка на ней левой кнопки мыши (рис. 4.17).

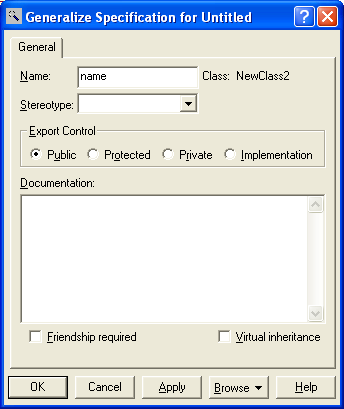


Рисунок 4.17 - Диалоговое окно спецификации свойств отношения обобщения

Для задания имени *обобщения* следует на единственной вкладке General (Общие) в поле ввода Name (Имя) ввести текст ее имени: следует и нажать кнопку Apply или OK, чтобы сохранить результаты редактирования имени *ассоциации*.

**3 Пример построения диаграммы классов**



Рисунок 4.18 – Диаграмма классов для алгоритма дискриминантного анализа

**Задание на лабораторное занятие**

1. Изучить теоретический материал
2. Согласно заданному варианту, разработайте диаграмму классов для реализации метода многомерного статистического анализа.

**Контрольные вопросы**

1. Каково назначение диаграммы классов?

2. Какими способами можно создать диаграмму?

3. Какие инструменты доступны для диаграммы?

4. Какие команды предоставляет контекстное меню класса?

5. Как настроить свойства атрибутов класса?

6. Как настроить свойства методов класса?

7. Какие типы отношений классов вы знаете?

# ЛАБОРАТОРНАЯ РАБОТА № 14-15

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

Цель работы: Подготовка модели к генерации программного кода; генерация программного кода

### **Теоретическая часть**

Одно из самых мощных свойств Rational Rose – возможность генерации программного кода, представляющего модель. Варианты генерации программ меняются в зависимости от установленной версии Rose. В настоящее время применяются следующие три различные версии:

- Rose Modeler – позволяет создавать модель системы, но не поддерживает генерацию программного кода и обратное проектирование.

- Rose Professional – позволяет генерировать программный код на одном языке.

- Rose Enterprise – позволяет генерировать программный код на Ada 83, Ada 95, ANSI C++, CORBA, Java, COM, Visual Basic, Visual C++, C++ и XML. Кроме того, поддерживается генерация кода и обратное проектирование баз данных.

Лабораторный практикум построен на изучении Case-средства Rational Rose версии - Enterprise Edition.

**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.

2.1.1 Проверка модели

В Rose 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: ToolsОписание: srarrCheck Model (ИнструментыОписание: srarrПроверить модель). 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. 5.1).

  
Рисунок 5.1- 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.

2.1.2 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 Visual Basic выполнять подобный шаг не обязательно. В Java и Visual Basic Rose создаст автоматически соответствующий компонент для каждого из классов.

Для создания компонента:

1. Откройте диаграмму Компонентов (Component).

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

2.1.3 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 Visual Basic этот шаг не является обязательным. Rose может генерировать программный код самостоятельно. При генерации в Rose программ Java и Visual Basic 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 Open Specification 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 (Logical view).

ИЛИ

1. Найдите класс в окне Logical view браузера.

2. Перетащите класс на нужный компонент в окне Component view.

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

2.1.4 Установка свойств генерации программного кода

Для каждого языка в Rose 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 Rose предлагаются общепринятые параметры по умолчанию.

2.1.4.1 Настройка свойств C++

Для того чтобы пользоваться возможностями C++, необходимо описать назначение этих свойств, список которых доступен во вкладке C++ спецификаций класса. (см. рис. 5.2)

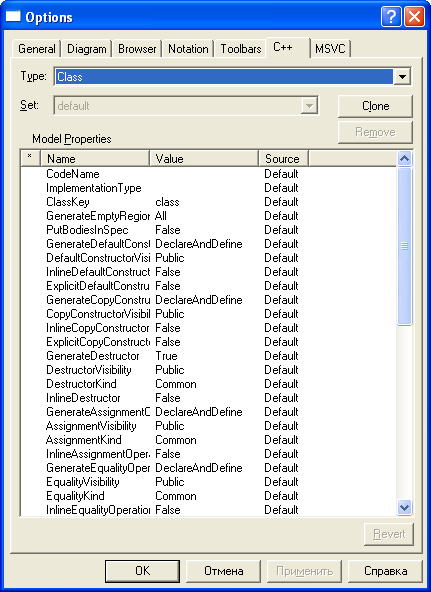


Рисунок 5.2 – Окно свойств генерации кода на С++

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 Rational Rose. 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 Rational Rose 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; Declare Only - 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. **Subscript Visibility** 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. **Dereference Visibility** - 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 »).

2.1.5 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.

2.1.6 Генерация программного кода

Если у вас установлены Rose Professional или Rose Enterprise, то в меню Tools предлагается несколько вариантов, специфичных для конкретного языка программирования (см. рис. 5.3).

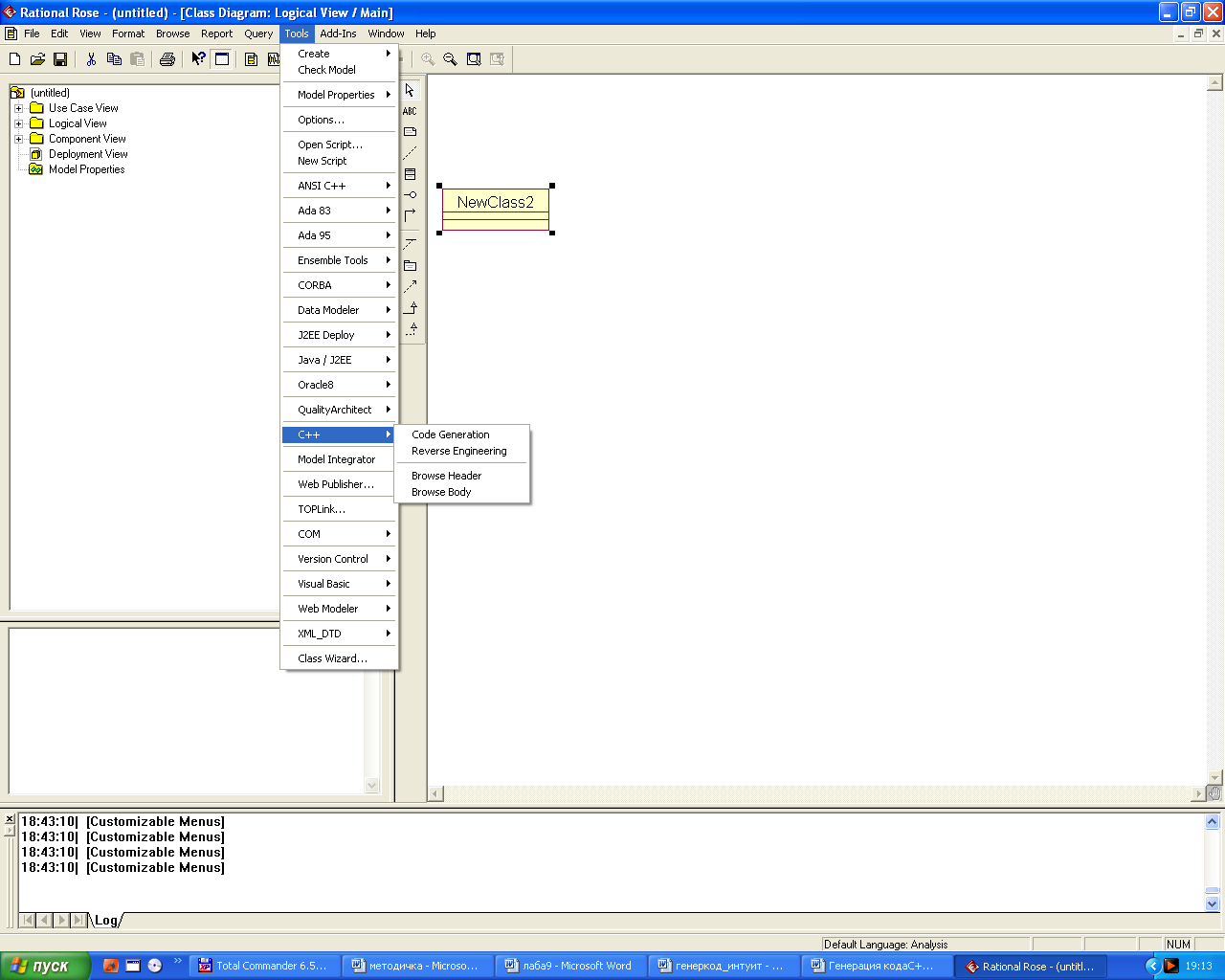


Рисунок 5.3 - Пункты меню генерации кода

Чтобы показать или скрыть эти пункты меню, выберите пункт Add-Ins → Add-Ins (Надстройки → Менеджер надстроек). В диалоговом окне Add-In Manager (см. рис. 5.4) с помощью флажков покажите или скройте нужные варианты для различных языков.

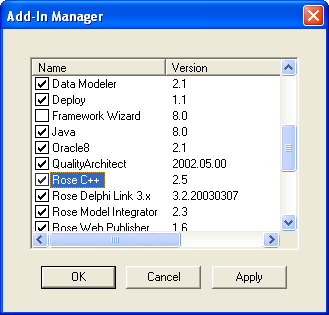


Рисунок 5.4 - Менеджер надстроек Add-Ins

Генерация программного кода в среде IBM Rational Rose 2003 возможна для отдельного класса или компонента. Для этого нужный элемент модели предварительно следует выделить в браузере проекта и выполнить операцию контекстного меню: Tools→C++→Code Generation - (Язык C++→Генерировать код). В результате этого будет открыто диалоговое окно с предложением выбора классов для генерации программного кода на выбранном языке программирования (рис. 5.5). После выбора соответствующих классов и нажатия кнопки OK программа IBM Rational Rose 2003 выполняет кодогенерацию.

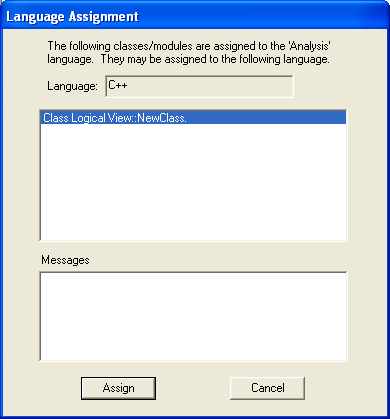


Рисунок 5.5 - Окно выбора классов для генерации программного кода

Затем происходит компиляция и выдается окно статуса (Code Generation Status). Здесь можно увидеть информацию о том, какой класс был закодирован и количество ошибок и предупреждений (рис. 5.6). Если у вас произошла, какая-либо ошибка или же предупреждение, то их можно увидеть на рабочем поле в Rational Rose, для этого и существует самое нижнее окно, в нем передаются все ваши действия и ошибки, произошедшие в ходе кодогенерации.

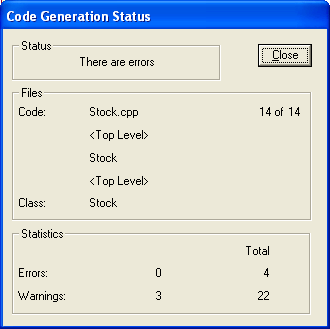


Рисунок 5.6 – Окно статуса компиляции

2.1.7 Результаты генерации

В результате кодогенерации Rational Rose 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 Browse Header и Browse Body, 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 Rose. 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 IBM Rational Rose 2003 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 IBM Rational Rose 2003 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 IBM Rational Rose 2003 и нотации языка UML для документирования и реализации соответствующих моделей.

**Задание на лабораторное занятие**

1. Изучить теоретический материал
2. Сгенерировать программный код на С++ для диаграммы классов, разработанной вами в предыдущей лабораторной работе.

**Содержание отчета**

- титульный лист;

- постановка задачи;

- листинг сгенерированного кода;

- вывод.

**Контрольные вопросы**

1. Какие диаграммы необходимо предварительно разработать, чтобы выполнить кодогенерацию?

2. Как посмотреть исходный код?

3. Какие установки свойств доступны на вкладке C++?

4. Какова структура создаваемого кода?

5. Что необходимо добавить в шаблоны классов для получения работоспособного приложения?

6. Какие шаги нужно предпринять для обновления модели по исходному коду?

7. Какие основные этапы кодогенерации вы знаете? Расскажите кратко о каждом из них?

# ЛАБОРАТОРНАЯ РАБОТА № 16

**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* (Component diagram) 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.

**Особенности разработки диаграмм компонентов в среде IBM Rational Rose 2003**

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* (Component View), 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 браузере (Component View) и дважды щелкнуть на пиктограмме Main (Главная).

Через пункт меню BrowseОписание: srarrComponent Diagram (БраузерОписание: srarrДиаграмма *компонентов*).

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. 6.1).

Таблица 6.1 - Назначение кнопок специальной панели инструментов диаграммы компонентов

|  |  |  |
| --- | --- | --- |
| **Графическое изображение** | **Всплывающая подсказка** | **Назначение кнопки** |
| Описание: 12_1 | Selection Tool | Превращает изображение курсора в форму стрелки для последующего выделения элементов на диаграмме |
| Описание: 12_2 | Text Box | Добавляет на диаграмму текстовую область |
| Описание: 12_3 | Note | Добавляет на диаграмму примечание |
| Описание: 12_4 | Anchor Note to Item | 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 | Subprogram Specification | Добавляет на диаграмму спецификацию подпрограммы |
| Описание: 12_9 | Subprogram Body | Добавляет на диаграмму тело подпрограммы |
| Описание: 12_10 | Main Program | Добавляет на диаграмму главную программу |
| Описание: 12_11 | Package Specification | Добавляет на диаграмму спецификацию пакета |
| Описание: 12_12 | Package Body | Добавляет на диаграмму тело пакета |
| Описание: 12_13 | Task Specification | Добавляет на диаграмму спецификацию задачи |
| Описание: 12_14 | Task Body | Добавляет на диаграмму тело задачи |
| Описание: 12_15 | Generic Subprogram | Добавляет на диаграмму типовую подпрограммы(по умолчанию отсутствует) |
| Описание: 12_16 | Generic Package | Добавляет на диаграмму типовой пакет (по умолчанию отсутствует) |
| Описание: 12_17 | Database | Добавляет на диаграмму базу данных (по умолчанию отсутствует) |

Программа IBM Rational Rose 2003 не поддерживает следующие графические стереотипы. Графическое изображение этих стереотипов и их краткая характеристика приводятся в следующей таблице (табл. 6.2). 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 6.2. Графическое изображение стереотипов компонентов и их характеристика

|  |  |  |
| --- | --- | --- |
| **Графическое изображение и имя по умолчанию** | **Название стереотипа** | **Характеристика стереотипа компонента** |
| Описание: 12_18 | Subprogram Specification | Спецификация подпрограммы. Содержит описание переменных, процедур и функций и не содержит определений классов |
| Описание: 12_19 | Subprogram Body | 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 | Main Program | Главная программа. Реализует базовую логику работы программного приложения и содержит ссылки на другие компоненты модели |
| Описание: 12_21 | Package Specification | Спецификация пакета. Содержит определение класса, его атрибутов и операций. В языке программирования С++ спецификации пакета соответствует отдельный файл с расширением «h» |
| Описание: 12_22 | Package Body | Тело пакета. Содержит код реализации операций класса. В языке программирования С++ спецификации пакета соответствует отдельный файл с расширением «cpp» |
| Описание: 12_23 | Task Specification | Спецификация задачи. Может содержать определение класса, его атрибутов и операций, которые предполагается использовать в независимом потоке управления |
| Описание: 12_24 | Task Body | Тело задачи. Может содержать реализацию операций класса, которые имеют независимый поток управления. |
| Описание: 12_25 | Generic Subprogram | 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 | Generic Package | 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 |

2.1.1 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. Добавить компонент на диаграмму можно также с помощью операции главного меню: ToolsОписание: srarrCreateОписание: srarrComponent или с помощью операции контекстного меню: NewОписание: srarrComponent, предварительно выделив представление компонентов в браузере проекта.

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. 6.1).

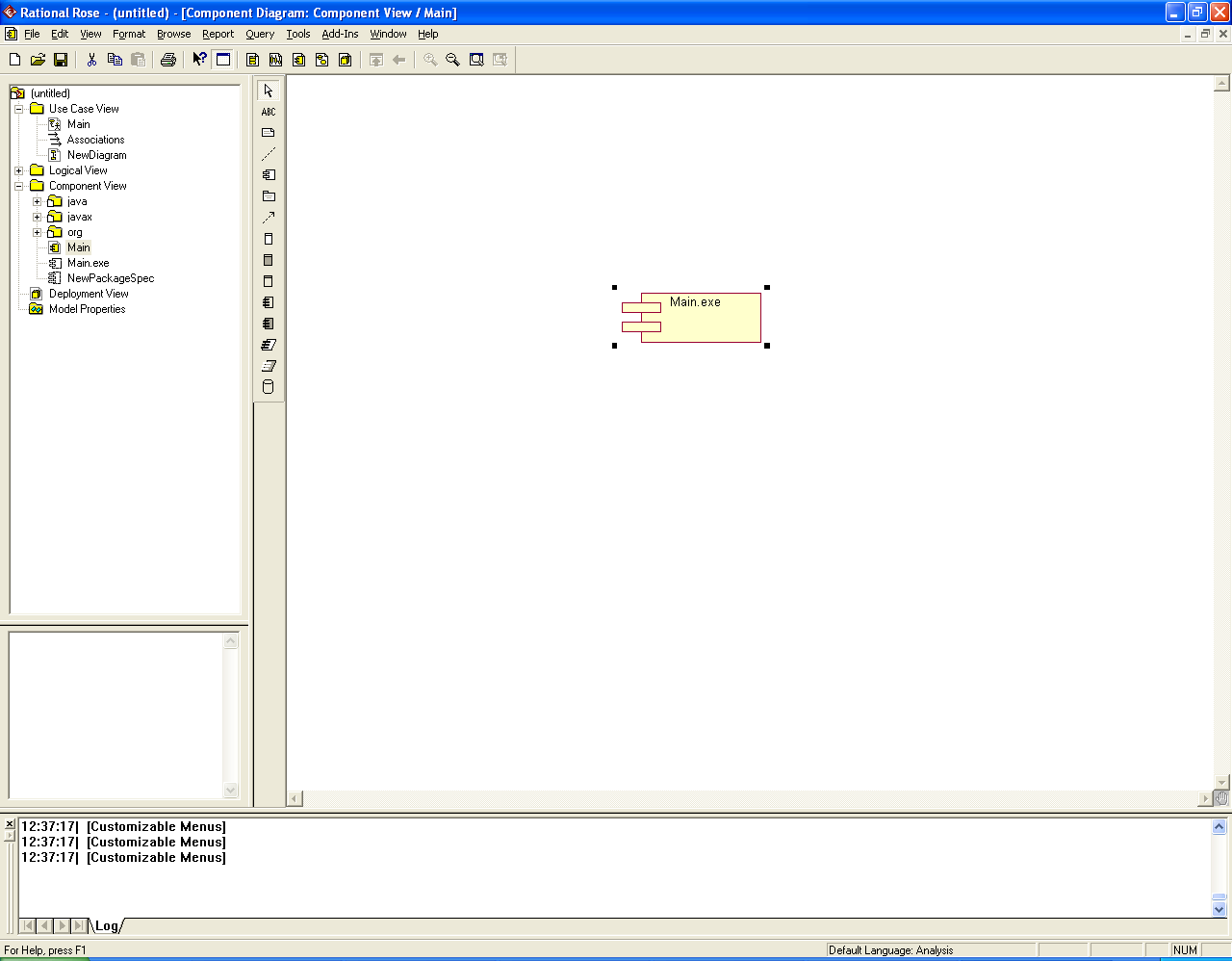


Рисунок 6.1 - Диаграмма компонентов после добавления компонента Main.exe

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. 6.2).

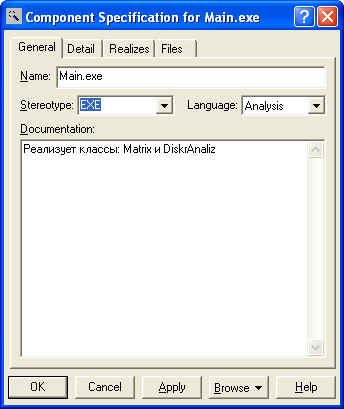


Рисунок 6.2 - Диалоговое окно спецификации свойств компонента Main.exe

В частности, для компонента Main.exe можно выбрать стереотип <<EXE>> из предлагаемого вложенного списка, поскольку применительно к разрабатываемой модели предполагается реализация этого компонента в форме исполнимого файла. При этом на вкладке Realizes (Реализует) содержатся все классы, включая и актеров, которые на данный момент присутствуют в модели.

По умолчанию в среде IBM Rational Rose 2003 для всех добавляемых на диаграмму компонентов в качестве языка реализации используется язык анализа, который в последствии следует изменить на тот язык программирования, который предполагается использовать для написания программного кода. В дальнейшем при генерации программного кода необходимо будет дополнительно выбрать те классы, которые реализует тот или иной компонент модели.

2.1.2 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.

2.1.3 Пример построения диаграммы компонентов

Ниже представлена диаграмма компонентов для программного средства, реализующего алгоритм дискриминантного анализа (рис. 6.3).



Рисунок 6.3 – Диаграмма компонентов рассматриваемого ПС

**2.2 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.

**Задание на лабораторное занятие**

1. Изучить теоретический материал
2. Выполнить тестирование и отладку информационной системы, разработанной в лабораторной работе 14-15, то есть разработать диаграмму компонентов рассматриваемого ПС.

**Содержание отчета**

- титульный лист;

- постановка задачи;

- тестовый набор данных и результаты тестирования;

- вывод.

# Варианты заданий

**2\_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\_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.

**2\_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.

**2\_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.

**2\_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.

**2\_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.

**2\_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.

**2\_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.

**2\_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.

**2\_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.

**2\_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.

**2\_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».

**2\_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.

**2\_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.

**2\_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.

**2\_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.

**2\_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).