Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
**«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»**

Кафедра информатики и защиты информации

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

Методические указания к выполнению курсовых, расчетно-практических работ

Составители:
Л.А. Артюшина
Ю.М. Монахов

Владимир 2012

УДК 004.056

Рецензент:
Доктор технических наук, профессор,
зав.кафедрой Информационных систем и информационного менеджмента
владимирского государственного университета
Александров Д.В.

Информационные технологии. Методические указания к выполнению курсовых, расчетно-практических работ / Л.А. Артюшина, Ю.М. Монахов; Владим. гос. ун-т. – Владимир: Издательство Владим. гос. ун-та, 2012. –

Ориентированы на изучение основных этапов объектно-ориентированного анализа и проектирования. Содержат темы, посвященные концептуальным основам объектно-ориентированной методологии разработки программных приложений, основным компонентам языка UML и отдельным видам канонических диаграмм, необходимым для представления информационной модели системы.

Предназначены для подготовки к выполнению курсовых и расчетно-проектных работ по дисциплинам «Информатика», «Информационные технологии» для студентов первого курса специальностей 210405-радиосвязь, радиовещание и телевидение, 140200 - электроэнергетика, 210200- проектирование и технология электронных средств, 200100 - приборостроение, 201000 - биотехнические системы и технологии.

ПРЕДИСЛОВИЕ

Методические указания предназначены для подготовки к выполнению курсовых и расчетно-проектных работ по дисциплинам «Информатика», «Информационные технологии».

Рекомендуется для студентов I курса специальностей 210405 («Радиосвязь, радиовещание и телевидение»), 140200 («Электроэнергетика»), 210200 («Проектирование и технология электронных средств»), 200100 («Приборостроение»), 201000 («Биотехнические системы и технологии»), изучающих объектно-ориентированные языки программирования типа C++ в рамках дисциплин «Информатика», «Информационные технологии».

Методические указания представлены в виде набора занятий, посвященных отдельным этапам объектно-ориентированного анализа и проектирования.

Так, в материалах занятия 1 кратко освещены концептуальные основы и история становления объектно-ориентированной методологии разработки программных приложений, основные компоненты языка UML с их краткой характеристикой.

Во втором и третьем занятии рассматриваются отдельные виды канонических диаграмм, необходимые для представления информационной модели системы.

Занятие 4 посвящено рассмотрению этапов разработки программ с использованием технологии объектно-ориентированного программирования.

В занятии 5 предложены варианты тем курсовых (расчетно-практических) работ, определено содержание курсовой (расчетно-практической) работы.

Изложение материала сопровождается примерами разработки диаграмм.

Глава 1. Введение

1.1 Методология структурного программирования

Двадцать и более лет назад программисты реализовывали свои проекты путем непосредственного написания кода (процедурные языки C, Pascal, FORTRAN). С возрастанием размера и сложности проектов становилось все яснее, что такой подход неудачен. Проблема заключалась в непропорциональном возрастании сложности процесса создания самих программ.

Со временем разработка больших программ превратилась в серьезную проблему и потребовала их разбиения на более мелкие фрагменты. Основой такого разбиения стала процедурная декомпозиция, при которой отдельные части программы или модули представляли собой совокупность процедур (функций) для решения некоторой совокупности задач.

Деление программы на функции и модули стало основой структурного программирования.

Недостатки структурного программирования:

- структурный подход не позволил в достаточной степени упростить большие сложные программы. Из-за своей сложности большие программы нередко содержат трудно обнаруживаемые ошибки. Ошибки в программном обеспечении потенциально могут стать причиной материального ущерба, а иногда и угрожать жизни людей (например, при управлении авиаполетами);

- разделение данных и функций, являющихся основой структурного подхода, плохо отражает картину реального мира. В реальном мире нам приходится иметь дело с физическими объектами (люди, машины). Эти объекты нельзя отнести ни к данным, ни к функциям, поскольку реальные вещи представляют собой совокупность *свойств* (характеристик) и *поведения*. Примерами свойств может быть цвет глаз, место работы, мощность двигателя, количество дверей. Поведение – это некоторая реакция объекта в ответ на внешнее воздействие. Если вы нажмете на тормоз автомобиля, это повлечет за собой остановку. Остановка – это пример поведения.

В результате борьбы с вышеперечисленными проблемами были выработаны три новые концепции программирования:

- Объектно-ориентированное программирование (ООП);

- Унифицированный язык моделирования (UML). Это графический язык, включающий в себя множество различных диаграмм, помогающих специалистам по системному анализу создавать алгоритмы, а программистам – разбираться в принципах работы программ;

– Специализированные средства разработки программного обеспечения.

1.2 Методология объектно-ориентированного программирования

Объектно-ориентированный подход к программированию стал приоритетным при разработке большинства программных проектов, так как предлагал новый мощный способ решения проблемы сложности программ, а именно, две ключевые концепции – *объекты* и *классы*. Под классом понимают некоторую абстракцию совокупности объектов, которые имеют общий набор свойств и обладают одинаковым поведением. Таким образом, класс является описанием совокупности сходных между собой объектов. Каждый объект рассматривается как экземпляр класса. Объекты, которые не имеют полностью одинаковых свойств или не обладают одинаковым поведением, по определению не могут быть отнесены к одному классу.

Важной особенностью классов является возможность их организации в виде некоторой иерархической структуры, что приводит к понятиям наследование, инкапсуляция, полиморфизм.

Принцип, в соответствии с которым каждый подкласс обладает свойствами, присущими тому классу, из которого он был выделен, называется наследованием.

Термин инкапсуляция характеризует сокрытие отдельных деталей внутреннего устройства классов от внешних по отношению к нему объектов или пользователей. Взаимодействующему с классом объекту или пользователю нет необходимости знать, каким образом реализован тот или другой метод класса, услугами которого он решил воспользоваться. Конкретная реализация присущих классу свойств и методов, которые определяют поведение этого класса, является собственным делом этого класса.

Под полиморфизмом понимают свойство некоторых объектов принимать различные внешние формы в зависимости от обстоятельств. Применительно к ООП полиморфизм означает, что действия, выполняемые одноименными методами, могут отличаться в зависимости от того к какому из классов относится тот или иной метод. Например, рассмотрим два объекта или класса: двигатель автомобиля и электрический свет в комнате. Для каждого из них можно определить операцию «выключить». Однако, сущность этой операции будет отличаться для каждого из рассмотренных объектов. Так, для двигателя автомобиля это будет означать прекращение подачи топлива и его остановку, а для электрического света в комнате это будет означать нажатие на кнопку выключателя. После чего в комнате станет темно.

Полиморфизм в ООП связан с перегрузкой, но не тождественен ей. Термин «перегрузка» характерен для ситуации, когда существующая операция, например = или +, наделяется возможностью совершать действия над операндами нового типа. В этом случае говорят, что такая операция является *перегруженной*.

1.3 Объектно-ориентированная декомпозиция

Широкое распространение методологии ООП оказало влияние на процесс разработки программ. Процедурная декомпозиция (ПД) уступила место объектно-ориентированной декомпозиции (ООД). Основное отличие ООД от ПД состоит в предварительной разработке концептуальной схемы, которая отражала бы общие взаимосвязи предметной области и особенности организации соответствующей информации. При этом под предметной областью принято понимать ту часть реального мира, которая имеет существенное или непосредственное отношение к процессу функционирования программы. Компоненты предметной области выбираются таким образом, чтобы при дальнейшей разработке их удобно было представлять в виде объектов и классов с соответствующими свойствами и методами. Таким образом, объектная декомпозиция предполагает определение:

1. сколько классов необходимо для решения задачи;
2. какие классы нужны для решения задачи;
3. какими свойствами должны обладать классы;
4. какими методами должны обладать классы;
5. характера взаимосвязей между классами.

Очевидно, что указанная совокупность задач связана не столько с написанием кода программы, сколько с анализом требований к будущей программе, а также с анализом конкретной предметной области, для которой разрабатывается программа. Все эти обстоятельства привели к появлению методологии объектно-ориентированного анализа и проектирования (ООАП).

Методология ООАП породила несколько подходов визуализации концептуальных схем, одним из которых является унифицированный язык моделирования (UML), который рассматривается в данном учебно-методическом пособии.

В рамках языка UML все представления о модели сложной системы фиксируются в виде специальных графических конструкций, получивших название диаграмм. В терминах языка UML определены следующие виды диаграмм:

1. Диаграмма вариантов использования.
2. Диаграмма классов.
3. Диаграмма состояний.

4. Диаграмма деятельности.
5. Диаграмма последовательности.
6. Диаграмма кооперации.
7. Диаграмма компонентов.
8. Диаграмма развертывания.

Для диаграмм языка UML существуют три типа визуальных обозначений, которые важны с точки зрения заключенной в них информации:

- связи, которые представляются различными линиями на плоскости;
- текст, который содержится внутри границ отдельных геометрических фигур на плоскости. При этом форма этих фигур соответствует некоторым элементам языка UML и имеет фиксированную семантику;
- графические символы, изображаемые от тех или иных визуальных элементов диаграмм.

При изображении диаграмм следует придерживаться следующих основных рекомендаций:

- каждая диаграмма должна служить законченным представлением соответствующего фрагмента моделируемой предметной области, т.е. в процессе разработки диаграммы необходимо учесть все компоненты, важные с точки зрения контекста данной модели и диаграммы;
- вся информация о компонентах должна быть явно представлена на диаграмме;
- диаграммы не должны содержать противоречивой информации;
- диаграммы не следует перегружать текстовой информацией;
- любая из моделей системы должна содержать только те элементы, которые определены в нотации языка UML;
- количество типов диаграмм для конкретной модели не является строго фиксированным. Однако, для представления сложных систем процесс ООАП следует разбить на отдельные этапы, на каждом из которых осуществить разработку соответствующих типов диаграмм модели системы. При этом последовательность разработки диаграмм совпадает с их последовательной нумерацией.

В данном учебном пособии мы ограничимся рассмотрением двух видов диаграмм, а именно, диаграммы классов и диаграммы вариантов использования.

Глава 2. Диаграмма вариантов использования

2.1 Вариант использования

Диаграмма вариантов использования описывает функциональное назначение системы, или другими словами то, что система будет делать в процессе своего функционирования. Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью вариантов использования. Кроме этого, с актерами могут быть связаны интерфейсы, которые определяют, каким образом другие элементы модели взаимодействуют с этими актерами. Диаграмма вариантов может дополняться пояснительным текстом, который раскрывает смысл или семантику составляющих ее компонентов. Такой пояснительный текст получил названия примечания или сценария.

Таким образом, структурно диаграмму вариантов использования можно представить как совокупность следующих элементов: варианты использования, актеры, интерфейсы и сценарии. Рассмотрим каждый элемент диаграммы подробнее.

Каждый *вариант использования* определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой.

Отдельный вариант использования обозначается на диаграмме эллипсом, внутри которого содержится его краткое название или имя в форме глагола с пояснительными словами (см. Рис 1).

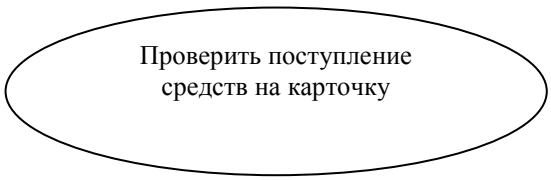

Рис.1

Каждый вариант использования соответствует отдельному сервису, который предоставляет система пользователю (актеру). Сервис представляет собой законченное действие, т.е. после того как система закончит обработку запроса пользователя, она должна вернуться в исходное состояние.

2.2 Актер

Актер представляет собой любую внешнюю по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач.

Стандартным графическим обозначением актера на диаграммах является фигурка "человечка", под которой записывается конкретное имя актера (см. Рис. 2).

Рис. 2

Имя актера должно быть достаточно информативным с точки зрения семантики.

Примерами актеров могут быть: клиент банка, банковский служащий, продавец магазина, менеджер отдела продаж, графический образ, водитель автомобиля, администратор гостиницы, сотовый телефон и другие сущности, имеющие отношение к концептуальной модели соответствующей предметной области.

Так как в общем случае актер всегда находится вне системы, его внутренняя структура никак не определяется. Для актера имеет значение только его внешнее представление, т. е. то, как он воспринимается со стороны системы.

Актеры взаимодействуют с системой посредством передачи и приема сообщений от вариантов использования.

Сообщение представляет собой запрос актером сервиса от системы и получение этого сервиса.

2.3 Интерфейсы

Интерфейс служит для спецификации параметров модели, которые видимы извне без указания их внутренней структуры. Применительно к диаграммам вариантов использования, интерфейсы определяют совокупность операций, которые обеспечивают необходимый набор сервисов или функциональности для актеров.

Интерфейсы содержат только операции без указания особенностей их реализации.

На диаграмме вариантов использования интерфейс изображается в виде маленького круга, рядом с которым записывается его имя (см. Рис. 3а). В качестве имени может быть существительное, которое характеризует соответствующую информацию или сервис (например, "датчик"), но чаще строка текста (например, "запрос к базе данных"). Если имя записывается на английском, то оно должно начинаться с заглавной буквы I, например, ISensor (см. Рис. 3 б).

Рис.3а

Рис.3б

Имена интерфейсов подчиняются общим правилам наименования компонентов языка UML. Имя может состоять из любого числа букв, цифр и некоторых знаков препинания, таких как двойное двоеточие "::". Последний

символ используется для более сложных имен, включающих в себя не только имя самого интерфейса (после знака), но и имя сущности, которая включает в себя данный интерфейс (перед знаком). Например, "Система аутентификации клиентов::форма ввода пароля".

Графический символ отдельного интерфейса может соединяться на диаграмме сплошной линией с тем вариантом использования, который его поддерживает. Сплошная линия в этом случае указывает на тот факт, что связанный с интерфейсом вариант использования должен реализовывать все операции, необходимые для данного интерфейса, а возможно и больше (см. Рис. 4). Кроме этого, интерфейсы могут соединяться с вариантами использования пунктирной линией со стрелкой (см. Рис. 4), означающей, что вариант использования предназначен для спецификации только того сервиса, который необходим для реализации данного интерфейса.

Рис. 4

В последующем интерфейс может быть уточнен явным указанием тех операций, которые специфицируют отдельный аспект поведения системы. В этом случае он изображается в форме прямоугольника класса с ключевым словом "interface" в секции имени, с пустой секцией атрибутов и с непустой секцией операций.

2.4 Примечания

Примечания в языке UML предназначены для включения в модель произвольной текстовой информации, имеющей непосредственное отношение к контексту разрабатываемого проекта. В качестве такой информации могут быть комментарии разработчика (например, дата и версия разработки диаграммы), ограничения (например, на значения отдельных связей). Применительно к диаграммам вариантов использования примечание может носить самую общую информацию, относящуюся к общему контексту системы.

Графически примечания обозначаются прямоугольником с "загнутым" верхним правым уголком (см. Рис.5). Внутри прямоугольника содержится текст примечания. Примечание может относиться к любому элементу диаграммы, в этом случае их соединяет пунктирная линия. Если примечание относится к нескольким элементам, то от него проводятся, соответственно, несколько линий.

Рис.5

2.5 Отношения на диаграмме вариантов использования

В языке UML имеется несколько стандартных видов отношений между актерами и вариантами использования:

- Отношение ассоциации.
- Отношение расширения.
- Отношение обобщения.
- Отношение включения.

Применительно к диаграммам вариантов использования оно служит для обозначения специфической роли актера в отдельном варианте использования.

Отношение ассоциации обозначается сплошной линией между актером и вариантом использования. Эта линия может иметь дополнительные условные обозначения, такие, например, как имя и кратность (см. Рис. 6).

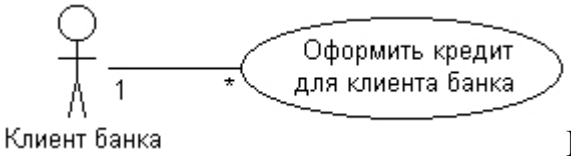

Рис. 6

Кратность характеризует общее количество конкретных экземпляров данного компонента, которые могут выступать в качестве элементов данной ассоциации.

Кратность ассоциации указывается рядом с обозначением компонента диаграммы, который является участником данной ассоциации. Применительно к диаграммам вариантов использования кратность имеет специальное обозначение в форме одной или нескольких цифр и специального символа "*" (звездочка).

Для диаграмм вариантов использования наиболее распространенными являются четыре основные формы записи кратности отношения ассоциации:

- целое неотрицательное число (включая цифру 0). Предназначено для указания кратности, которая является строго фиксированной для элемента соответствующей ассоциации. В этом случае количество экземпляров актеров или вариантов использования, которые могут

выступать в качестве элементов отношения ассоциации, в точности равно указанному числу;

- два целых неотрицательных числа, разделенные двумя точками и записанные в виде: "первое число .. второе число". Данная запись в языке UML соответствует нотации для множества или интервала целых чисел, которая применяется в некоторых языках программирования для обозначения границ массива элементов. Эту запись следует понимать как множество целых неотрицательных чисел, следующих в последовательно возрастающем порядке;
- два символа, разделенные двумя точками. При этом первый из них является целым неотрицательным числом или 0, а второй — специальным символом "*". Здесь символ "*"обозначает произвольное конечное целое неотрицательное число, значение которого неизвестно на момент задания соответствующего отношения ассоциации;
- если кратность отношения ассоциации не указана, то по умолчанию принимается ее значение, равное 1.

Отношение расширения определяет взаимосвязь экземпляров отдельного варианта использования с более общим вариантом, свойства которого определяются на основе способа совместного объединения данных экземпляров. Отношение расширения является направленным и указывает, что применительно к отдельным примерам некоторого варианта использования должны быть выполнены конкретные условия, определенные для расширения данного варианта использования. Так, если имеет место отношение расширения от варианта использования A к варианту использования B, то это означает, что свойства экземпляра варианта использования B могут быть дополнены благодаря наличию свойств у расширенного варианта использования A.

Отношение расширения между вариантами использования обозначается пунктирной линией со стрелкой, направленной от того варианта использования, который является расширением для исходного варианта использования. Данная линия со стрелкой помечается ключевым словом "extend" ("расширяет"), как показано на рис. 7.

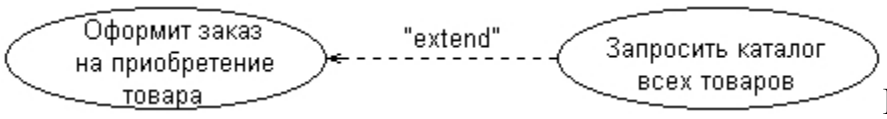

Рис.7

Данное отношение включает в себя некоторое условие и ссылки на точки расширения в базовом варианте использования. Чтобы расширение имело место, должно быть выполнено определенное условие данного отношения. Ссылки на точки расширения определяют те места в базовом варианте использования, в которые должно быть помещено соответствующее расширение при выполнении условия.

В представленном выше примере (см. Рис. 7) при оформлении заказа на приобретение товара только в некоторых случаях может потребоваться

предоставление клиенту каталога всех товаров. При этом условием расширения является запрос от клиента на получение каталога товаров. Очевидно, что после получения каталога клиенту необходимо некоторое время на его изучение, в течение которого оформление заказа приостанавливается. После ознакомления с каталогом клиент решает либо в пользу выбора отдельного товара, либо отказа от покупки вообще. Сервис или вариант использования "Оформить заказ на приобретение товара" может отреагировать на выбор клиента уже после того, как клиент получит для ознакомления каталог товаров (см. Рис.8).

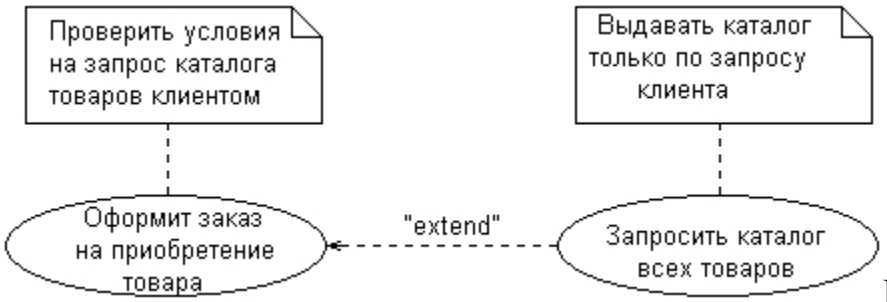

Рис.8

Отношение обобщения служит для указания того факта, что некоторый вариант использования A может быть обобщен до варианта использования B. В этом случае вариант A будет являться специализацией варианта B. При этом B называется предком или родителем по отношению A, а вариант A — потомком по отношению к варианту использования B. Следует подчеркнуть, что потомок наследует все свойства и поведение своего родителя, а также может быть дополнен новыми свойствами и особенностями поведения. Графически данное отношение обозначается сплошной линией со стрелкой в форме незакрашенного треугольника, которая указывает на родительский вариант использования (см. Рис.9). Эта линия со стрелкой имеет специальное название - стрелка "обобщение".

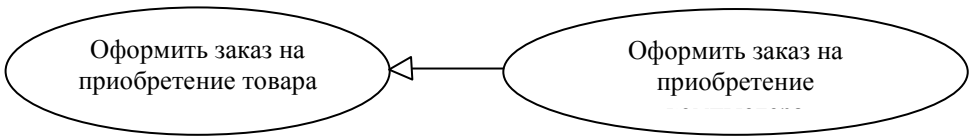

Рис.9

Один вариант использования может иметь несколько родительских вариантов. В этом случае реализуется множественное наследование свойств и поведения отношения предков. С другой стороны, один вариант использования может быть предком для нескольких дочерних вариантов.

Отношение включения между двумя вариантами использования указывает, что некоторое заданное поведение для одного варианта использования включается в качестве составного компонента в последовательность поведения другого варианта использования.

Семантика этого отношения определяется следующим образом. Когда экземпляр первого варианта использования в процессе своего выполнения

достигает точки включения в последовательность поведения экземпляра второго варианта использования, экземпляр первого варианта использования выполняет последовательность действий, определяющую поведение экземпляра второго варианта использования, после чего продолжает выполнение действий своего поведения. При этом предполагается, что даже если экземпляр первого варианта использования может иметь несколько включаемых в себя экземпляров других вариантов, выполняемые ими действия должны закончиться к некоторому моменту, после чего должно быть продолжено выполнение прерванных действий экземпляра первого варианта использования в соответствии с заданным для него поведением.

Один вариант использования может быть включен в несколько других вариантов, а также включать в себя другие варианты. Включаемый вариант использования может быть независимым от базового варианта в том смысле, что он предоставляет последнему некоторое инкапсулированное поведение, детали реализации которого скрыты от последнего и могут быть легко перераспределены между несколькими включаемыми вариантами использования. Более того, базовый вариант может зависеть только от результатов выполнения включаемого в него поведения, но не от структуры включаемых в него вариантов.

Графически данное отношение обозначается пунктирной линией со стрелкой, направленной от базового варианта использования к включаемому. При этом данная линия со стрелкой помечается ключевым словом "include" ("включает"), как показано на рис. 10.

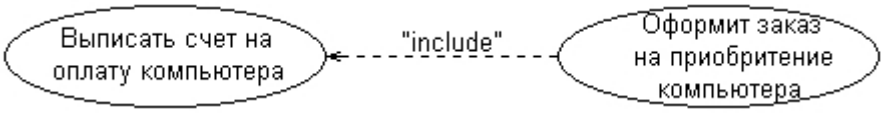

Рис. 10

Глава 3. Диаграмма классов

3.1 Класс

Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о временных аспектах функционирования системы. С этой точки зрения диаграмма классов является дальнейшим развитием концептуальной модели проектируемой системы. Основным элементом диаграмм этого типа является класс.

Как уже отмечалось во введении, класс в языке UML служит для обозначения множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов.

Графически класс изображается в виде прямоугольника (см. Рис.1а, б, в), который дополнительно может быть разделен горизонтальными линиями на разделы или секции. В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы).

а б в Рис.1

3.2 Имя класса

Обязательным элементом обозначения класса является его имя. На начальных этапах разработки диаграммы отдельные классы могут обозначаться простым прямоугольником с указанием только имени соответствующего класса (Рис.1а). По мере проработки отдельных компонентов диаграммы описания классов дополняются атрибутами (Рис.1б) и операциями (Рис.1в).

Предполагается, что окончательный вариант диаграммы содержит наиболее полное описание классов, которые состоят из трех разделов или секций. Иногда в обозначениях классов используется дополнительный четвертый раздел, в котором приводится семантическая информация справочного характера или явно указываются исключительные ситуации. Даже если секция атрибутов или операций является пустой, в обозначении класса она выделяется горизонтальной линией, чтобы сразу отличить класс от других элементов языка UML (см. Рис.2).

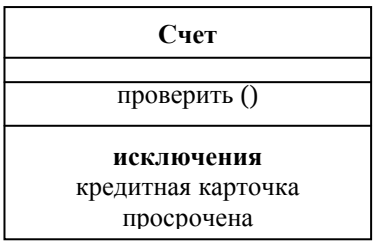

Рис.2

Имя класса должно быть уникальным в пределах пакета, который описывается некоторой совокупностью диаграмм классов (возможно, одной диаграммой). Оно указывается в первой верхней секции прямоугольника. Имя класса записывается по центру секции имени полужирным шрифтом и должно начинаться с заглавной буквы. Рекомендуется в качестве имен классов использовать существительные, записанные без пробелов.

Если необходимо явно указать к какому пакету относится тот или иной класс используется специальный символ разделитель — двойное двоеточие "::". Синтаксис строки имени класса в этом случае будет следующий <Имя пакета>::<Имя класса>.

В первой секции обозначения класса могут находиться ссылки на стандартные шаблоны или абстрактные классы, от которых образован данный класс и, соответственно, от которых он наследует свойства и методы. В этой секции может приводиться информация о разработчике данного класса и статус состояния разработки, а также могут записываться и другие общие свойства этого класса, имеющие отношение к другим классам диаграммы или стандартным элементам языка UML.

Примерами имен классов могут быть такие существительные, как «Сотрудник», «Компьютер», «Круг», «Компания» и т.п., имеющие непосредственное отношение к моделируемой предметной области и функциональному назначению проектируемой системы.

Класс может не иметь экземпляров или объектов. В этом случае он называется абстрактным классом, а для обозначения его имени используется наклонный шрифт (курсив). В языке UML принято общее соглашение о том, что любой текст, относящийся к абстрактному элементу, записывается курсивом.

3.3 Атрибуты класса

Во второй сверху секции прямоугольника класса записываются его *атрибуты* или свойства.

Каждому атрибуту класса соответствует отдельная строка текста, которая состоит из квантора видимости атрибута, имени атрибута, его кратности, типа значений атрибута и, возможно, его исходного значения:
<квантор видимости><имя атрибута>[кратность]: <тип атрибута> = <исходное значение>{строка-свойство}

Рассмотрим указанные характеристики подробнее.

Квантор видимости может принимать одно из трех возможных значений и, соответственно, отображается при помощи специальных символов:

– символ "+" обозначает атрибут с областью видимости типа общедоступный (public).
– символ "#" обозначает атрибут с областью видимости типа защищенный (protected).
– знак "-" обозначает атрибут с областью видимости типа закрытый (private).

Квантор видимости может быть опущен. В этом случае его отсутствие просто означает, что видимость атрибута не указывается. Эта ситуация отличается от принятых по умолчанию соглашений в традиционных языках программирования, когда отсутствие квантора видимости трактуется как public или private. Однако вместо условных графических обозначений можно записывать соответствующее ключевое слово: public, protected, private.

Имя атрибута используется в качестве идентификатора соответствующего атрибута и поэтому должно быть уникально в пределах данного класса. Имя атрибута является единственным обязательным элементом синтаксического обозначения атрибута.

Кратность атрибута характеризует общее количество конкретных атрибутов данного типа, входящих в состав отдельного класса. В общем случае кратность записывается в форме строки текста в квадратных скобках после имени соответствующего атрибута: *[нижняя граница1 .. верхняя граница1, нижняя граница2.. верхняя грашца2, ..., нижняя граница k .. верхняя граница k],* где нижняя и верхняя границы являются положительными целыми числами, каждая пара которых служит для обозначения отдельного замкнутого интервала целых чисел. В качестве верхней границы может использоваться специальный символ "*", который означает произвольное положительное целое число.

Значения кратности из интервала следуют в монотонно возрастающем порядке без пропуска отдельных чисел, лежащих между нижней и верхней границами. При этом придерживаются следующего правила: соответствующие нижние и верхние границы интервалов включаются в значение кратности. Если в качестве кратности указывается единственное число, то кратность атрибута принимается равной данному числу. Если же указывается единственный знак "*", то это означает, что кратность атрибута может быть произвольным положительным целым числом или нулем.

В качестве примера рассмотрим следующие варианты задания кратности атрибутов.

– [0..1] означает, что кратность атрибута может принимать значение 0 или 1. При этом 0 означает отсутствие значения для данного атрибута.
– [0..*] означает, что кратность атрибута может принимать любое положительное целое значение большее или равное 0.
– [1..*] означает, что кратность атрибута может принимать любое положительное целое значение большее или равное 1.
– [1..5] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 4, 5.

- [1..3,5,7] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 5, 7.
- [1..3,7.. 10] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 7, 8, 9, 10.
- [1..3,7..*] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, а также любое положительное целое значение большее или равное 7.
- если кратность атрибута не указана, то по умолчанию принимается ее значение равное 1.

Тип атрибута в простейшем случае указывается строкой текста, имеющей осмысленное значение в пределах пакета или модели, к которым относится рассматриваемый класс.

Примерами задания имен и типов атрибутов классов могут быть:

- цвет: Color — здесь цвет является именем атрибута, Color — именем типа данного атрибута. Указанная запись может определять традиционно используемую RGB-модель для представления цвета. В этом случае имя типа Color как раз и характеризует семантическую конструкцию, которая применяется в большинстве языков программирования для представления цвета.
- форма: Многоугольник — здесь имя атрибута форма может характеризовать такой класс, который является геометрической фигурой на плоскости. В этом случае тип атрибута Многоугольник указывает на тот факт, что отдельная геометрическая фигура может иметь форму треугольника, прямоугольника, ромба, пятиугольника и любого другого многоугольника, но не окружности или эллипса.

Строка-свойство служит для указания значений атрибута, которые не могут быть изменены в программе при работе с данным типом объектов. Фигурные скобки обозначают фиксированное значение соответствующего атрибута для класса в целом, которое должны принимать все вновь создаваемые экземпляры класса без исключения. Отсутствие строки-свойства по умолчанию трактуется так, что значение соответствующего атрибута может быть изменено в программе.

Например, строка-свойство в записи атрибута: *заработная плата: Currency = = {$500}* может служить для обозначения фиксированной заработной платы для каждого объекта класса "Сотрудник" определенной должности в некоторой организации. С другой стороны, запись данного атрибута в виде: *заработная плата: Currency = $500* означает уже нечто иное, а именно — при создании нового экземпляра Сотрудник (например, при приеме на работу нового сотрудника) для него устанавливается по умолчанию заработная плата в $500. Однако для отдельных сотрудников могут быть сделаны исключения, как в большую, так и в меньшую сторону, о чем необходимо позаботиться дополнительно в программе.

3.4 Методы класса

В третьей сверху секции прямоугольника записываются *операции или методы класса*. Операция представляет собой некоторый сервис, предоставляющий каждый экземпляр класса по определенному требованию. Совокупность операций характеризует функциональный аспект поведения класса.

Каждой операции класса соответствует отдельная строка, которая состоит из квантора видимости операции, имени операции, выражения типа возвращаемого операцией значения и, возможно, строки-свойства данной операции: *<квантор видимости><имя операции>(список параметров):<выражение типа возвращаемого значения>{строка-свойство}*

Так как указанные характеристики аналогичны характеристикам атрибутов класса, остановимся только на некоторых из них.

Выражение типа возвращаемого значения является зависимой от языка реализации спецификацией типа или типов значений параметров, которые возвращаются объектом после выполнения соответствующей операции. Двоеточие и выражение типа возвращаемого значения могут быть опущены, если операция не возвращает никакого значения.

Строка-свойство служит для указания значений свойств, которые могут быть применены к данному элементу. Строка-свойство не является обязательной, она может отсутствовать, если никакие свойства не специфицированы.

По умолчанию под областью операции понимается объект класса. Операция с областью действия на весь класс показывается подчеркиванием имени и строки выражения типа возвращаемого значения.

Для повышения производительности системы одни операции могут выполняться параллельно или одновременно, а другие — только последовательно. В этом случае для указания параллельности выполнения операции используется строка-свойство вида "{concurrency = имя}", где имя может принимать одно из следующих значений:

- *последовательная (sequential)* — для данной операции необходимо обеспечить ее единственное выполнение в системе, одновременное выполнение других операций может привести к ошибкам или нарушениям целостности объектов класса;
- *параллельная (concurrent)* — данная операция в силу своих особенностей может выполняться параллельно с другими операциями в системе, при этом параллельность должна поддерживаться на уровне реализации модели;
- *охраняемая (guarded) — все* обращения к данной операции должны быть строго упорядочены во времени с целью сохранения целостности объектов данного класса, при этом могут быть приняты дополнительные меры по конт*ролю исключительных* ситуаций на этапе ее выполнения.

Отсутствие данной строки-свойства означает, что семантика параллельности для операции не определена и данная операция требует последовательного выполнения.

Появление сигнатуры операции на самом верхнем уровн*е* объявляет эту операцию на весь класс, при этом данная операция наследуется всеми потомками данного класса. Если в некотором классе операция не выполняется (т. е. некоторый метод не применяется), то такая операция может быть помечена как абстрактная "{abstract}". Другой способ показать абстрактный характер операции — записать ее сигнатуру курсивом. В качестве примеров записи операций можно привести следующие обозначения отдельных операций:

- +*создать* () — может обозначать абстрактную операцию по созданию отдельного объекта класса, которая является общедоступной и не содержит формальных параметров. Эта операция не возвращает никакого значения после своего выполнения;
- +нарисовать (форма: Многоугольник = прямоугольник, цвет заливки: Color = (0, 0, 255)) — может обозначать операцию по отображению на экране монитора прямоугольной области синего цвета;
- выдать сообщение ():{"Ошибка деления на ноль"} — смысл данной операции не требует пояснения, поскольку содержится в строке-свойстве операции. Данное сообщение может появиться на экране монитора в случае попытки деления некоторого числа на ноль, что недопустимо.

3.5 Отношения между классами на диаграмме классов

Кроме структуры классов на соответствующей диаграмме указываются различные отношения между классами. Базовыми отношениями или связями в языке UML являются:

- отношение зависимости;
- отношение ассоциации;
- отношение агрегации;
- отношение обобщения.

Каждое из этих отношений имеет собственное графическое представление на диаграмме, которое отражает взаимосвязи между объектами соответствующих классов.

Отношение зависимости используется в такой ситуации, когда некоторое изменение одного элемента модели может потребовать изменения другого зависимого от него элемента модели.

Отношение зависимости графически изображается пунктирной линией между соответствующими элементами со стрелкой на одном из ее концов. На диаграмме классов данное отношение связывает отдельные классы между собой, при этом стрелка направлена от класса-клиента зависимости к независимому классу или классу-источнику (см. Рис. 3).

Рис.3

На данном рисунке класс В является источником некой зависимости, класс А – клиентом этой зависимости.

Отношение ассоциации связывает разнотипные, не похожие друг на друга классы. Такие классы взаимодействуют между собой в процессе выполнения каких-либо операций. Ассоциации могут иметь имя, направления и роли, изображаются графически символом треугольника в форме стрелки. Например, в системе учета студентов некоторого института классы Институт и Студент ассоциированы между собой (см. Рис.4), причем Институт выступает в роли «Обучающая организация», Студент - в роли «Обучаемый».

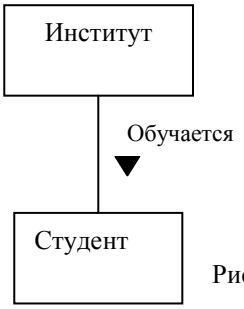

Рис.4

Для ассоциации также можно определить характеристики множественности, уточняющей, сколько классов может участвовать в одной связи с каждой стороны. Допускается указывать следующие характеристики:

- от 0 до бесконечности, т.е. любое целое положительное число;
- целое .. * - от заданного числа до бесконечности;
- целое – точное определение количества классов;
- целое1, целое2 – несколько вариантов точного количества классов;
- целое1 .. целое2 – диапазон классов.

Отношение агрегации имеет место между несколькими классами в том случае, если один из классов представляет собой некоторую сущность, включающую в себя в качестве составных частей другие независимые сущности. Графически отношение агрегации изображается сплошной линией, один из концов которой представляет собой незакрашенный ромб (см. Рис.5).

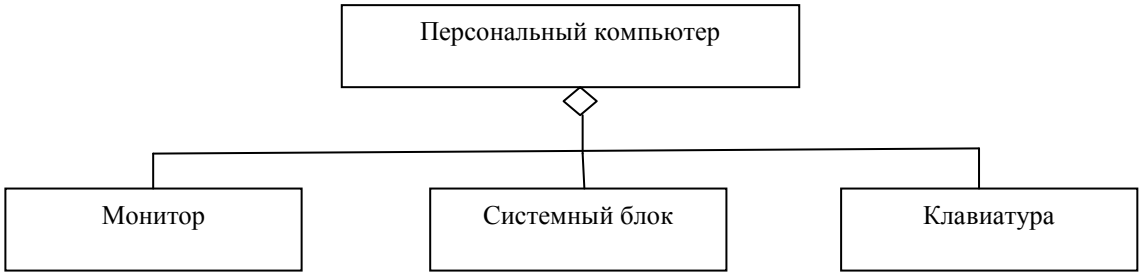

Рис.5

Отношение обобщения имеет место между однотипными, имеющими общие признаки, но все же разными понятиями, при котором одно из понятий является обобщающим и включает в себя другое понятия. Применительно к диаграмме классов данное отношение описывает иерархическое строение классов и наследование их свойств и поведения. При этом предполагается, что класс-потомок обладает всеми свойствами и поведением класса-предка, а также имеет свои собственные свойства и поведение, которые отсутствуют у класса-предка. На диаграммах обобщения изображаются сплошной линией с треугольной стрелкой.

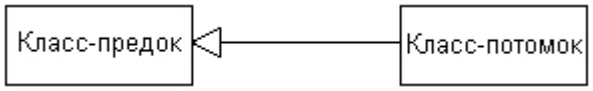

Рис.6

Рядом со стрелкой обобщения может размещаться строка текста, указывающая на некоторые дополнительные свойства этого отношения. Отмеченное свойство касается всех подклассов данного отношения. Если текст следует рассматривать как ограничение, тогда он записывается в фигурных скобках.

В качестве ограничений могут быть использованы следующие ключевые слова языка UML:

- {complete} — означает, что в данном отношении обобщения специфицированы все классы-потомки, и других классов-потомков у данного класса-предка быть не может. Пример - класс Клиент банка является предком для двух классов: Физическое_лицо и Компания, и других классов-потомков он не имеет. На соответствующей диаграмме классов это можно указать явно, записав рядом с линией обобщения данную строку-ограничение;
- {disjoint} — означает, что классы-потомки не могут содержать объектов, одновременно являющихся экземплярами двух или более классов. В приведенном ниже рисунке 7 это условие выполняется, поскольку предполагается, что Прямоугольник не может являться одновременно Геометрической фигурой и Окружностью, например. В этом случае рядом с линией обобщения можно записать данную строку-ограничение;
- {incomplete} — означает случай, противоположный первому. А именно, предполагается, что на диаграмме указаны не все классы-

потомки. В последующем возможно восполнить их перечень не изменяя уже построенную диаграмму. Пример — диаграмма класса "Геометрическая фигура" (см. Рис.7), для которой указание всех без исключения видов геометрических фигур соизмеримо с созданием соответствующего каталога. С другой стороны, для частной задачи в этом нет необходимости. Но указать неполноту структуры классов-потомков все же следует;

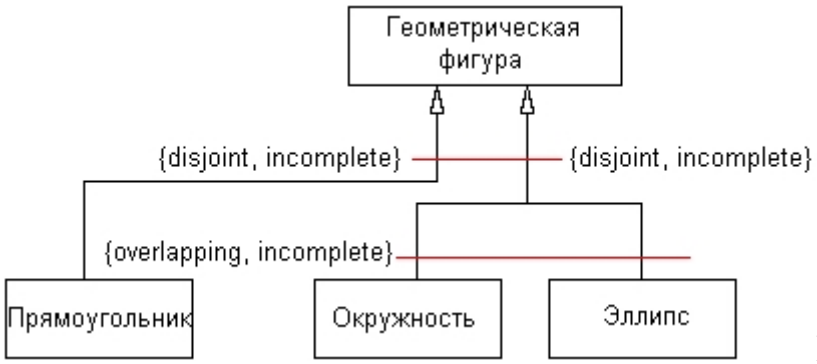

Рис.7

– {overlapping} — означает, что отдельные экземпляры классов-потомков могут принадлежать одновременно нескольким классам. Пример - класс "Геометрическая фигура" является классом-предком для класса "Окружность" и класса "Эллипс". Однако существует отдельный класс "Дуга", экземпляры которого одновременно являются объектами первых двух классов. Вполне естественно такую ситуацию указать явно с помощью данной строки-ограничения.

Глава 4. Этапы разработки программ с использованием объектно-ориентированного программирования (ООП)

4.1 Объектно-ориентированный анализ

Процесс разработки программ с использованием ООП включает четыре этапа: объектно-ориентированный анализ, проектирование, реализация (объектно-ориентированное программирование), модификация. В рамках пособия рассмотрим первых три этапа.

В процессе объектно-ориентированного анализа основное внимание уделяется определению и описанию:

1. объектов (понятий) в терминах предметной области;
2. важнейших особенностей поведения объектов (понятий) и сообщений, передаваемых между ними.

Результат анализа выражается в модели предметной области.

Таким образом, модель предметной области – это представление понятий, выраженных в терминах предметной области задачи.

Модель предметной области иллюстрируется набором диаграмм вариантов использования, созданных в процессе объектно-ориентированного проектирования. На этом этапе определяются программные объекты и способы их взаимодействия с целью выполнения системных требований.

Помимо динамического представления взаимосвязи объектов, отображаемой на диаграммах вариантов использования, на этом этапе строят статическое представление системы в виде диаграммы классов. На такой диаграмме отображаются атрибуты и методы классов. В отличие от модели предметной области эта диаграмма не иллюстрирует понятия реального мира, а описывает программные классы.

На этапе объектно-ориентированного программирования обеспечивается реализация разработанных компонентов на языке C++ , Java и т.п.

Модификация – это процесс добавления новых функциональных возможностей или изменение существующих свойств системы в зависимости от изменяющихся условий эксплуатации.

4.2 Проектирование

Покажем, как осуществлять разработку программы с использованием ООП на примере программы, имитирующую карточную игру. Один игрок (компьютер) показывает вам три карты, затем перемешивает их и раскладывает. Необходимо угадать на каком месте, например, стала 1 карта?

1 этап. Разрабатываем модель предметной области задачи. Для этого выделяем объекты (понятия). В нашем примере используются три объекта: Карты, Пользователь, Монитор (управляющий объект) (см. Рис.1).

2 шаг. Для иллюстрации важных этапов игры и передачи сообщений между объектами строим еще одну диаграмму вариантов использования (см. Рис.2).

Игра начинается по инициативе пользователя, поэтому можно сказать,

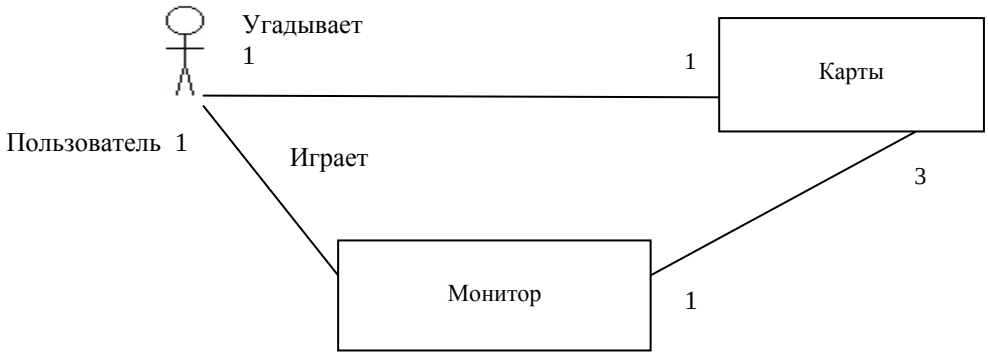

Рис.1 Модель предметной области игры в карты

что объект Пользователь посылает сообщение «Играет» объекту Монитор. В свою очередь объект Монитор посылает сообщение «Вызывает» объекту Карты. В ответ на это сообщение объект Карты с помощью своего метода выводит на экран все карты согласно их масти и достоинству. В реальном мире три карты выбирает и тасует игрок, в программе эту функцию выполняет объект Монитор. Пользователь угадывает одну из трех карт. В случае правильного ответа объект Монитор выводит сообщение «Вы угадали», в противном случае – «Вы не угадали».

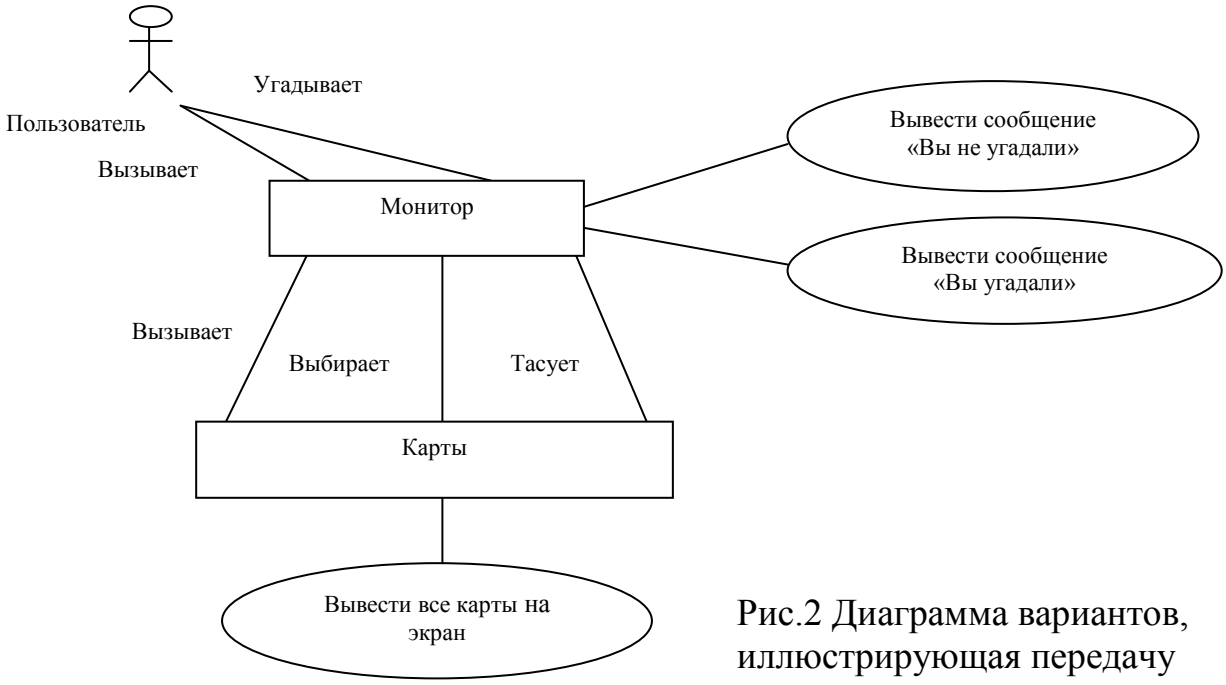

Рис.2 Диаграмма вариантов, иллюстрирующая передачу сообщений между программными

Анализ диаграммы взаимодействия игры (Рис.2) приводит к диаграмме классов, показанной на рис.3.

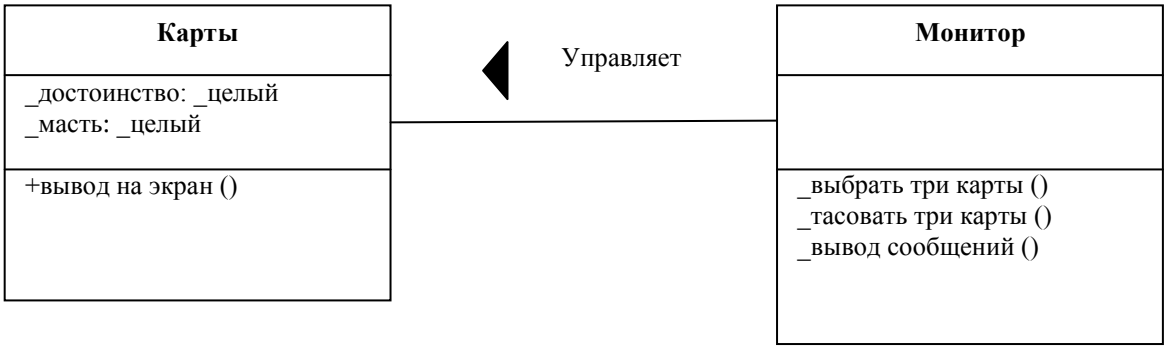

Рис.3

4.3. Объектно-ориентированное программирование

3 шаг. Объектно-ориентированное программирование.

Создаем класс card (поля – масть и достоинство), 1 метод – выводящий на экран карты согласно их достоинству и масти.

Определяем 4 объекта класса card: 3 карты и 1 вспомогательный объект для перетасовывания карт.

```
//card.cpp
#include <iostream.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
class card
{
private:
   int dost;
   int mast;
public:
void k (int a, int b)
{
  dost=a;
  mast=b;
 }
void m (int a, int b)
{
if (a>=2 && a<=10)
 cout <<a;
else
   switch (a)
  {
   case 11: cout<<"valet"; break;
   case 12: cout<<"dama"; break;
   case 13: cout<<"korol"; break;
   case 14: cout<<"tuz"; break;
    }
    switch (b)
```

```
  {
   case 1: cout<<" bubn\n"; break;
   case 2: cout<<" chervi\n"; break;
   case 3: cout<<" piki\n"; break;
   case 4: cout<<" trefi\n"; break;
   }
   }
};
void main ()
{
clrscr();
int posizion;
card k1, k2, k3, temp;
cout<<"3 karti\n";

// выбираем случайным образом 3 карты с помощью функции rand.
srand ((unsigned)(time (NULL)));
int a1=rand() % 14+1;
int b1=rand() % 5+1;
cout<<a1<<" "<<b1;

//полям класса присваиваются значение масти и достоинства
k1.k(a1,b1);
k2.k(a1,b1);
k3.k(a1,b1);
k1.m(a1,b1);
k2.m (a1,b1);
k3.m(a1,b1);

// компьютер тасует карты
temp = k1;
k1= k2;
k2 = k3;
k3 = temp;

//игрок угадывает карту
cout<<"na kakoj pozizi 1 karta?";
cin>>posizion;
switch (posizion)
{
 case 1: cout<<"now"; break;
 case 2: cout<<"now"; break;
 case 3: cout<<"yes"; break;
};
getch (); }
```

Задания
на курсовую (расчетно-практическую) работу
по дисциплинам
«Информатика», «Информационные технологии»
тема
«Технология объектно-ориентированного программирования»

Разработайте программы с использованием технологии ООП. Оформите отчет по курсовой (расчетно-практической) работе.

Содержание отчета:

- 1 лист - титульный лист с указанием темы курсовой работы и индивидуальной тематики;
- 2 лист - содержание с указанием страниц;
- 3 лист - индивидуальное задание курсовой работы согласно варианту (см. таблицу 1);
- 4(5) лист(ы) – объектная декомпозиция задачи, проиллюстрированная набором диаграмм (диаграмма вариантов использования, контекстная диаграмма классов);
- 5 (6, 7, 8) лист(ы) – листинг программы с комментариями;
- 6 (9) лист – список используемой литературы.

Таблица 1

Варианты заданий

№ варианта	Текст задания
1	«Игра в кости»: игрок бросает два кубика. Если сумма очков равна семи, участник считается победителем, в противном случае – проигравшим.
2	«Записная книжка»: в удобной для пользователя форме позволяет записывать и затем находить телефоны различных людей. Удобная форма для данной программы предполагает общение программы с пользователем через интерфейс типа «Меню». Набор возможных операций: открытие записной книжки, добавление записи, поиск записи по полю «фамилия».
3	Игра «Бридж»: в игре «Бридж» каждому из игроков раздают 13 карт, таким образом, колода расходуется полностью. Напишите программу «Игра в Бридж» в которой колода карт сначала перемешивается, затем делится на четыре части по 13 карт каждая. Каждая из четырех групп карт затем должна быть выведена на

	экран.
4	Игра «Угадай число»: составьте программу определения компьютером задуманного человеком числа от 1 до 1000 с помощью 10 вопросов. Ответы человека – да, нет.
5	Игра «Слово»: дано слово «телегазета». Составить программу, которая проверяет любое введенное пользователем слово и выдает сообщение, можно ли его составить из букв, входящих в заданное слово. Каждую букву можно использовать не больше числа ее вхождений в заданное слово.
6	Программа «Деканат»: составьте программу назначения стипендии студентам по результатам сессии, используя следующие правила: - если все оценки - 5, назначается повышенная стипендия; - если все оценки – 4 и 5, назначается обычная стипендия; - если есть оценка 3, стипендия не назначается. В программе должен быть предусмотрен ввод фамилии, имени студента и его оценок не менее, чем по четырем дисциплинам. В результате работы программы должен быть напечатан список группы с оценками и средними баллами каждого студента и два списка фамилий (назначенных на повышенную и обычную стипендию).
7	Игра «Кости». Смоделировать игру в кости двух игроков. На каждом ходе игрок бросает три игральных кубика.
8	Написать программу игры «Чет или нечет». Игрок пытается угадать четное или нечетное число появиться на экране. Угадывание проводится до тех пор, пока в ответ на запрос «Продолжить еще раз?» будет введено «Нет». В результате игры определяется количество верных и неверных ответов.
9	«Телефонный справочник»: в удобной для пользователя форме позволяет вводить в любом порядке фамилии и соответствующие телефоны, сортирует введенные записи в алфавитном порядке, позволяет вводить (удалять) новые записи так, чтобы алфавитный порядок не нарушился. Удобная форма для данной программы

	предполагает общение программы с пользователем через интерфейс типа «Меню». Набор возможных операций: формирование телефонного справочника, открытие телефонного справочника, добавление новой записи, удаление новой записи, поиск записи по полю «фамилия», поиск записи по полю «телефон».
10	Программа «Пункт проката». Известна информация о 30 клиентах пункта проката: фамилия, имя, отчество, адрес, домашний телефон, если он есть, название (я) предметов, взятых им в прокат. Предусмотреть в программе: -вывод всех клиентов пункта проката; - вывод клиентов, взявших в прокат определенную вещь; -реализовать общение программы с работником пункта через интерфейс типа «Меню».
11	Программа «Касса кинотеатра». Предусмотреть в программе вывод информации: - о стоимости билетов в зависимости от ряда (1-5 ряд – 400 руб., 6-11 ряд – 500 руб., с 12 ряда -600 руб., балкон – 300 руб.); -о продаже билета на то или иное место.
12	Программа «Расписание»: содержит информацию о дне недели, количестве пар, времени начала и конца пар, названии предметов, фамилии преподавателей для одной группы. С помощью программы вывести информацию: -о расписании группы на определенный день недели; -по фамилии преподавателя – его расписание; -по названию дисциплины – дни недели и время ее проведения.
13	Программа «Экзаменационная ведомость». Известна информация о предмете, номере группы, номерах зачетных книжек, фамилиях, именах и отчествах студентов. Используя эти данные, в удобной для пользователя форме, осуществить: -подсчет количества отличников, хорошистов, троечников и двоечников;

	-вывод оценок по полю «фамилия»; -вывод оценок по полю «номер зачетной книжки»; -вывод успеваемости группы по полю «дисциплина». Удобная форма для данной программы предполагает общение программы с пользователем через интерфейс типа «Меню». Набор возможных операций задать самостоятельно, исходя из условия задачи.
14	Игра «Секретный замок». Секретный замок для сейфа состоит из 10 расположенных в ряд ячеек, в которые надо вставить игральные кубики. Дверь открывается только в том случае, если в любых трех соседних ячейках сумма точек на передних гранях кубиков равна 10 (игральный кубик имеет на каждой грани от 1 до 6 точек). Напишите программу, которая разгадывает код замка при условии, что два кубика уже вставлены в ячейки. Во время работы программы на экран выводятся протокол поиска и результат.

ОГЛАВЛЕНИЕ

БИБЛИОГРАФИЯ

1. Абрамов, С.А. Начала информатики /С.А. Абрамов, Е.В. Зима. – М.: Наука, 1989. – 256 с.
2. Иванова, Г.С. Объектно-ориентированное программирование: Учебник для вузов. – 3-е изд., стер. / Г. С. Иванова, Т.Н. Ничушкина, Е.К. Пугачев / Под ред. Г.С. Ивановой. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2007. – 368 с. – ISBN 978-5-7038-2775-8.
3. Информатика. Программирование: С++: учеб.пособие. в 2 ч. Ч.1. Введение в язык С++/ Л.А. Артюшина, Ю.М. Монахов, А.А. Воронин; Владим.гос.ун-т. – Владимир: Изд-во Владим.гос.ун-та, 2011. – 132 с. – ISBN 978-5-9984-0124-4.
4. Ларман, К. Применение UML и шаблонов проектирования. – 2-е изд.: Пер. с англ. / К. Ларман. – М.: Издательский дом «Вильямс», 2002. – 624 с.- ISBN 5-8459-0250-9.
5. Леоненков, А.В. Самоучитель UML / А.В. Леоненков. – СПб.: БХВ-Петербург, 2007. – 576 с. - ISBN 978-5-94157-878-8