УВАЖАЕМЫЕ СТУДЕНТЫ! Законспектируйте в своей рабочей тетради по дисциплине приведенную лекцию (объемом 4-5 страницы), ответьте *письменно* на контрольные вопросы.

Результаты работы, фотоотчет, предоставить преподавателю на e-mail: r.bigangel@gmail.com до 30.01.2023.

При возникновении вопросов по приведенному материалу обращаться по следующему номеру телефона: (072)111-37-59, (Viber, WhatsApp), vk.com: https://vk.com/daykini

ВНИМАНИЕ!!! При отправке работы, не забывайте указывать ФИО студента, наименование дисциплины, дата проведения занятия (по расписанию).

Лекция 27

Тема: Объектно-ориентированное программирование.

Цель: Изучить существующие подходы к программированию и сущность объектно-ориентированного программирования.

Вопросы лекции:

1. *Другие подходы к разработке ПС*
2. *Объектно-ориентированный подход к программированию*
   - 2.1. *Объекты и отношения в программировании. Сущность объектного подхода к разработке программных средств.*
   - 2.2. *Особенности объектного подхода к разработке внешнего описания программного средства.*
   - 2.3. *Особенности объектного подхода на этапе конструирования программного средства.*
   - 2.4. *Особенности объектного подхода на этапе кодирования программного средства.*

1. Другие подходы к разработке ПС

Одним из важнейших признаков классификации языков программирования является принадлежность их к одному из стилей, основными из которых являются следующие стили:

- процедурный,
- функциональный,
- логический
- объектно-ориентированный.

Каждому стилю программирования, если следовать строгому методологическому подходу соответствует свой подход к организации жизненного цикла ПС.

Процедурное программирование

Процедурное (императивное) программирование является отражением архитектуры традиционных ЭВМ, которая была предложена фон Нейманом в 40-х годах. Теоретической моделью процедурного программирования служит алгоритмическая система под названием «машина Тьюринга».

Программа на процедурном языке программирования состоит из последовательности операторов (инструкций), задающих процедуру решения задачи. Основным является оператор присваивания, служащий для изменения содержимого областей памяти. Концепция памяти как хранилища значений, содержимое которого может обновляться операторами программы, является фундаментальной в императивном программировании.

Выполнение программы сводится к последовательному выполнению операторов с целью преобразования исходного состояния памяти, то есть значений исходных данных, в заключительное, то есть в результаты. Таким образом, с точки зрения программиста имеются

программа и память, причем первая последовательно обновляет содержимое последней.

Процедурные языки характеризуются следующими особенностями:

- необходимостью явного управления памятью, в частности, описанием переменных;
- малой пригодностью для символьных вычислений;
- отсутствием строгой математической основы;
- высокой эффективностью реализации на традиционных ЭВМ.

Одним из важнейших классификационных признаков *процедурного* языка является его уровень. Уровень языка программирования определяется семантической (смысловой) емкостью его конструкций и степенью его ориентации на программиста. Язык программирования частично ликвидирует разрыв между методами решения различного рода задач человеком и вычислительной машиной. Чем более язык ориентирован на человека, тем выше его уровень. Дадим краткую характеристику реализованным на ПЭВМ языкам программирования в порядке возрастания их уровня.

Язык Ассемблера — это язык**,** предназначенный для представления в удобочитаемой символической форме программ, записанных на машинном языке**.** Он позволяет программисту пользоваться мнемоническими кодами операций, присваивать удобные имена ячейкам и областям памяти, а также задавать наиболее удобные схемы адресации.

Язык Макроассемблера является расширением языка Ассемблера путем включения в него макросредств. С их помощью в программе можно описывать последовательности инструкций с параметрами — макроопределения. После этого программист может использовать снабженные аргументами макрокоманды, которые в процессе ассемблирования программы автоматически замещаются макрорасширениями. Макрорасширение представляет собой макроопределение с подставленными вместо параметров аргументами.

Другими словами, язык Макроассемблера представляет средства определения и использования новых, более мощных команд как последовательности базовых инструкций, что несколько повышает его уровень.

Языки Ассемблера и Макроассемблера применяются системными программистами - профессионалами с целью использования всех возможностей оборудования ЭВМ и получения эффективной по времени выполнения и по требуемому объему памяти программы. На этих языках обычно разрабатываются относительно небольшие программы, входящие в состав системного программного обеспечения: драйверы, утилиты и другие.

Язык программирования С (Си) первоначально был разработан для реализации операционной системы UNIX в начале 70-х годов. В последующем приобрел высокую популярность среди системных и прикладных программистов. В настоящее время этот язык реализован на большинстве ЭВМ.

В С сочетаются достоинства современных высокоуровневых языков в части управляющих конструкций и структур данных с возможностями доступа к аппаратным средствам ЭВМ на уровне, который обычно ассоциируется с языком низкого уровня типа языка Ассемблера. Язык С имеет синтаксис, обеспечивающий краткость программы, а компиляторы способны генерировать эффективный объектный код.

Одна из наиболее существенных особенностей С состоит в нивелировании различий между выражениями и операторами, что приближает его к функциональным языкам. В частности, выражение может обладать побочным эффектом присваивания, а также может использоваться в качестве оператора. Нет также четкой границы между процедурами и функциями, более того, понятие процедуры не вводится вообще.

Синтаксис языка затрудняет программирование и восприятие составленных программ. Отсутствует и строгая типизация данных, что предоставляет дополнительные возможности программисту, но не способствует написанию надежных программ.

Basic (Beginners All-purpose Symbolic Instruction Code) — многоцелевой язык символических инструкций для начинающих) представляет собой простой язык программирования, разработанный в 1964 году для использования новичками. Он был разработан как простейший язык для непосредственного общения человека с вычислительной машиной. Поэтому первоначально работа велась в интерактивном режиме с использованием интерпретаторов.

В настоящее время для этого языка имеются также и компиляторы. Согласно концепциям, заложенным в Basic, этот язык в смысле строгости и стройности является антиподом языка Pascal. В частности, в нем широко распространены различные правила умолчания, что считается плохим тоном в большинстве языков программирования подобного типа. Basic широко распространен на ЭВМ различных типов и очень популярен в среде программистов, особенно начинающих. Существует множество диалектов этого языка, мало совместимых между собой. Basic активно поглощает многие концепции и новинки из других языков. Поэтому он достаточно динамичен, и нельзя однозначно определить его уровень.

Pascal является одним из наиболее популярных среди прикладных программистов процедурным языком программирования, особенно для ПЭВМ. Разработанный в 1970 году швейцарским специалистом в области вычислительной техники профессором Н. Виртом, язык назван в честь французского математика и по замыслу автора предназначался для обучения программированию. Однако язык получился настолько удачным, что стал одним из основных инструментов прикладных и системных программистов при решении задач вычислительного и информационно-логического характера. В 1979 году был подготовлен проект описания языка — Британский стандарт языка программирования Pascal BS6192, который стал также и международным стандартом ISO 7185.

В языке Pascal реализован ряд концепций, рассматриваемых как основа «дисциплинированного» программирования и заимствованных впоследствии разработчиками многих языков. Одним из существенных признаков языка Pascal является последовательная и достаточно полная реализация концепции структурного программирования. Причем это осуществляется не только путем упорядочивания связей между фрагментами программы по управлению, но и за счет структуризации данных. Кроме того, в языке реализована концепция определения новых типов данных на основе уже имеющихся. Этот язык, в отличие от языка С, является строго типизированным. Pascal характеризуется:

- высоким уровнем;
- широкими возможностями;
- стройностью, простотой и краткостью;
- строгостью, способствующей написанию эффективных и надежных программ;
- высокой эффективностью реализации на ЭВМ.

Pascal реализован на ЭВМ различных типов, но наиболее распространен и развит для ПЭВМ. В настоящее время широко используются такие версии этого языка для ПЭВМ, как Borland Pascal и Turbo Pascal.

Функциональное программирование

Сущность функционального (аппликативного) программирования определена А. П. Ершовым как «... способ составления программ, в которых единственным действием является вызов функции, единственным способом расчленения программы па части является введение имени для функции, а единственным правилом композиции — оператор суперпозиции функции. Никаких ячеек памяти, ни операторов присваивания, ни циклов, ни, тем более, блок-схем, ни передачи управления».

Роль основной конструкции в функциональных языках играет выражение. К выражениям относятся скалярные константы, структурированные объекты, функции, тела функций и вызовы функций. Функция трактуется как однозначное отображение из X^ **в X,** где X — множество выражений.

Аппликативный язык программирования включает следующие **элементы:**

- классы констант, которыми могут манипулировать функции;
- набор базовых функций, которые программист может использовать **без** предварительного объявления и описания;
- правила построения новых функций из базовых;
- правила формирования выражений на основе вызовов функций.

Программа представляет собой совокупность описаний функций и выражения, которые необходимо вычислить. Данное выражение вычисляется посредством редукции, то есть серии упрощений, до тех пор, пока это возможно по следующим правилам: вызовы базовых функций

заменяются соответствующими значениями; вызовы небазовых функций заменяются их телами, в которых параметры замещены аргументами.

Функциональное программирование не использует концепцию памяти как хранилища значений переменных. Операторы присваивания отсутствуют, вследствие **чего** переменные обозначают не области памяти, а объекты программы, что полностью соответствует понятию переменной в математике. В принципе, можно составлять программы и вообще без переменных. Кроме того, нет существенных различий между константами и функциями, то есть между программами и данными. В результате этого функция может быть значением вызова другой функции и может быть элементом структурированного объекта. Число аргументов при вызове функции **не** обязательно должно совпадать с числом параметров, указанных при ее описании. Перечисленные свойства характеризуют аппликативные языки как языки программирования очень высокого уровня.

Первым таким языком был LISP (LIST Processing — обработка списков), созданный в 1959 году. Цель его создания состояла в организации удобства обработки символьной информации. Существенная черта этого языка — унификация программных структур и структур данных: все выражения записываются в виде списков.

Логическое программирование

Новую область — логическое, или реляционное программирование, — открыло появление языка PROLOG {Пролог} (PROgramming in LOGic — программирование в терминах логики). Этот язык был создан французским ученым А. Кольмероэ в 1973 году. В настоящее время известны и другие языки, однако наиболее развитым и распространенным языком логического программирования является именно Пролог. Так, имеется свыше 15 различных его реализации на ПЭВМ. Языки логического программирования, в особенности Пролог, широко используются в системах искусственного интеллекта.

Центральным понятием в логическом программировании является отношение. Программа представляет собой совокупность определений отношений между объектами (в терминах условий или ограничений) и цели (запроса). Процесс выполнения программы трактуется как процесс общезначимости логической формулы, построенной из программы по правилам, установленным семантикой используемого языка. Результат вычисления является побочным продуктом этого процесса. В реляционном программировании нужно только специфицировать факты, **на** которых алгоритм основывается, а не определять последовательность шагов, которые требуется выполнить. Это свидетельствует о декларативности языка логического программирования. Она метко выражена в формуле Р. Ковальского: «алгоритм = логика + управление». Языки логического программирования характеризуются:

- высоким уровнем;
- строгой ориентацией на символьные вычисления;
- возможностью инверсных вычислений, то есть переменные в процедурах не делятся на входные и выходные;
- возможной логической неполнотой, поскольку зачастую невозможно выразить в программе определенные логические соотношения, а также невозможно получить из программы все выводы правильные.

Логические программы, в принципе, имеют небольшое быстродействие, так как вычисления осуществляются методом проб и ошибок, поиском с возвратами к предыдущим шагам.

2. Объектно-ориентированный подход к программированию

2.1. Объекты и отношения в программировании. Сущность объектного подхода к разработке программных средств.

Окружающий нас мир состоит из объектов и отношений между ними. Согласно В.Далю объект (предмет) – это все, что представляется чувствам (объект вещественный) или уму (объект умственный). Таким образом, *объект* воплощает некоторую сущность и имеет некоторое состояние, которое изменяется со временем как следствие влияния других объектов, находящихся с первым в каких-либо отношениях. Он может иметь внутреннюю структуру: состоять из других

объектов, также находящихся между собой в некоторых отношениях. Исходя из этого, можно построить иерархическое строение мира из объектов. Однако при каждом конкретном рассмотрении окружающего нас мира некоторые объекты считаются неделимыми, причем в зависимости от целей рассмотрения такими (неделимыми) могут приниматься объекты разного уровня иерархии.

Отношение связывает некоторые объекты: можно считать, что объединение этих объектов обладает некоторым свойством. Если отношение связывает n объектов, то такое отношение называется n-местным (n-арным). На каждом месте объединения объектов, которые могут быть связаны каким-либо конкретным отношением, могут находиться разные объекты, но вполне определенные (в этом случае говорят: объекты определенного класса). Одноместное отношение называется *простым свойством объекта* (соответствующего класса). Многоместное отношение объектов будем называть *ассоциативным свойством объекта*, если этот объект участвует в этом отношении. Состояние объекта может быть изучено по значению простых или ассоциативных свойств этого объекта. Множество всех объектов, которые обладают каким-то общим набором свойств, называется *классом объектов*.

В процессе познания или изменения окружающего нас мира мы всегда принимаем в рассмотрение ту или иную упрощенную модель мира (*модельный мир*), в которую включаем объекты и отношения некоторых интересующих нас классов из окружающего нас мира. Каждый объект, имеющий внутреннюю структуру, может представлять свой модельный мир, включающий объекты этой структуры и отношения, которые их связывают. Таким образом, окружающий нас мир, можно рассматривать (в некотором приближении) как иерархическую структуру модельных миров.

В настоящее время в процессе познания или изменения окружающего нас мира широко используется компьютерная техника для обработки различного рода информации. В связи с этим применяется компьютерное (информационное) представление объектов и отношений. Каждый объект информационно может быть представлен некоторой структурой данных, отображающей его состояние. Простые свойства этого объекта могут задаваться непосредственно в виде отдельных компонент этой структуры, либо специальными функциями над этой структурой данных. Ассоциативные свойства (n-местные отношения для n>1) можно представить либо в активной форме, либо в пассивной форме. В активной форме n-местное отношение представляется некоторым программным фрагментом, реализующим либо n-местную функцию (определяющую значение свойства соответствующего объединения объектов), либо процедуру, осуществляющую изменение состояний некоторых из них. В пассивной форме такое отношение может быть представлено некоторой структурой данных (в которую могут входить и представления объектов, связываемых этим отношением), интерпретируемую на основании принятых соглашений по общим процедурам, независящим от конкретных отношений (например, реляционная база данных). В любом случае представление отношения определяет некоторые действия по обработке данных.

При исследовании модельного мира пользователи могут по-разному получать информацию от компьютера.

В одних случаях пользователей может интересовать получение информации об отдельных свойствах определенных объектов или результаты какого-либо взаимодействия между некоторыми объектами модельного мира. Для удовлетворения таких запросов разрабатываются соответствующие ПС, которые выполняют интересующие пользователей функции, или подходящие информационные системы, способные выдавать информацию об интересующих пользователей отношениях. В начальный период развития компьютерной техники (при не достаточно высокой мощности компьютеров) такой подход к исследованию модельного мира был вполне естественным. Именно он и провоцировал *реляционный* (*процедурный*, *функциональный* и т.п.) подход к разработке ПС, который был подробно рассмотрен в предшествующих лекциях. Сущность этого подхода состоит в систематическом использовании *декомпозиции функций* (*отношений*) для описания и построения структуры ПС (включая тексты программ). При этом сами объекты модельного мира, с которыми связаны заказываемые и реализуемые функции, представлялись фрагментарно (в том объеме, который необходим для выполнения этих функций)

и в форме, удобной для реализации этих функций. Тем самым обеспечивалась эффективная реализация требуемых функций, но не создавалось цельного и адекватного компьютерного представления модельного мира, интересующего пользователя. Попытки даже незначительного расширения объема и характера информации об этом модельном мире, которую можно получить от ПС, могло потребовать серьезной модернизации этого ПС.

В других случаях пользователя может интересовать наблюдение за изменением состояний объектов модельного мира в результате их взаимодействий. Это требует использования подходящих информационных моделей таких объектов, создания программных средств, моделирующих процессы взаимодействия объектов модельного мира, и предоставление пользователю доступа к этим информационным моделям (к *пользовательским* объектам). С помощью традиционных методов разработки это оказалось довольно трудоемкой задачей.

Наиболее полно отвечает решению этой задачи *объектный* подход к разработке ПС. Сущность его состоит в систематическом использовании *декомпозиции объектов* при описании и построении ПС. При этом функции (отношения), выполняемые таким ПС, будут выражаться через отношения объектов других уровней, т.е. их декомпозиция будет существенно зависеть от декомпозиции объектов.

С точки зрения разработчиков ПС следует различать следующие категории объектов (и, соответственно, их классов):

- объекты модельного (вещественного или умственного) мира,
- информационные модели объектов реального мира (будем называть их *пользовательскими объектами*),
- объекты процесса выполнения программ,
- объекты процесса разработки ПС (*технологические объекты программирования*).

Кроме того, в зависимости от способа представления в компьютере модельного мира и характера взаимодействия с ним со стороны пользователя следует различать пассивные и активные объекты.

Пассивный объект представляет собой некоторый фрагмент информационной среды, который способен хранить разные данные определенного типа (представляющие разные *состояния* этого объекта) и с которым связан некоторый набор операций (*применимых* к этому объекту). Операции над таким объектом применяются под воздействием некоторой *внешней* по отношению к этому объекту *активной силы*, исходящей либо от пользователя, либо от какого-либо программного фрагмента в процессе его выполнения.

Активный объект представляет собой такое расширение пассивного объекта, в котором фрагмент информационной среды способен также хранить и программные фрагменты, способные находиться в процессе выполнения (в *активном состоянии*). Активный объект, у которого какие-либо программные фрагменты находятся в активном состоянии, способен воспринимать сообщения или сигналы из операционной среды, в которую он погружен, и самостоятельно выполнять некоторые операции как реакцию на эти сообщения или сигналы. Таким образом, можно считать, что активный объект обладает *внутренней активной силой*.

Когда говорят об объектно-ориентированном подходе к разработке ПС, имеют в виду объектный подход с ориентацией на описание объектов модельного мира и построением их информационных моделей, причем используются, в основном, активные объекты. При этом многие процессы разработки ПС приобретают специфические («объектные») черты:

- использование системы понятий, позволяющих описывать объекты и их классы,
- декомпозиция объектов является основным средством упрощения ПС,
- использование внепрограммных абстракций для упрощения процессов разработки,
- предпочтение (приоритет) разработки структуры данных перед реализацией функций.

Основные из этих специфических особенностей разработки ПС покажем в рамках водопадной модели технологии.

2.2. Особенности объектного подхода к разработке внешнего описания программного средства.

При объектном подходе этап внешнего описания ПС оказывается существенно более емким и содержательным по сравнению с процедурным подходом.

Определение требований заказчика заключается в неформальном описании модельного мира, который пользователь собирается изучать или просто использовать с помощью требуемого ПС. При этом повышается роль прототипирования, которое при этом подходе часто окупается уменьшением объема работы на последующих этапах разработки ПС. Часто определение требований завершается разработкой *диаграммы вариантов использования(use case diagram)*, в которой фиксируются, в каких случаях будет использоваться ПС. Это позволяет при разработке ПС ограничиться только такими его функциональными возможностями, которые поддерживают эти случаи (варианты) использования. По существу, диаграмма вариантов использования представляет собой описание некоторого контекста использования (см. лекцию 4) *Неправильное понимание требований заказчиком, пользователями и разработчиками связано обычно с различными взглядами на роль требуемого ПС в среде его использования. Поэтому важной задачей при создании определения требований является установление контекста использования ПС, включающего связи между этим ПС, аппаратурой и людьми. Лучше всего этот контекст в определении требований представить в графической форме (в виде диаграмм) с добавлением описаний сущностей используемых объектов (блоков ПС, компонент аппаратуры, персонала и т.п.) и характеристики связей между ними.*

На рис.1. приведен пример диаграммы вариантов использования.

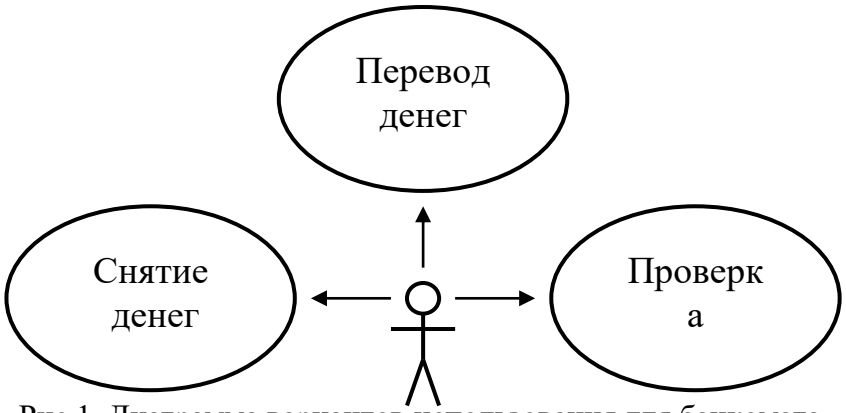

Рис.1. Диаграмма вариантов использования для банкомата.

Существенно изменяется содержание процесса спецификации требований: вместо разработки функциональной спецификации ПС создается формальное описание модельного мира, состоящее из трех частей:

- объектной модели (объектная модель определяет то, с чем что-то случается);
- динамической модели (динамическая модель определяет, когда это случается;
- функциональной модели (функциональная модель определяет то, что случается).

Объектная модель показывает статическую объектную структуру модельного мира, который должно представлять разрабатываемое ПС (программная система). Она включает определения используемых классов объектов и отношений между этими классами, а также определение используемых объектов этих классов и отношения между этими объектами.

Обычно *класс объектов* в объектной модели представляется в виде тройки (Имя класса, Список атрибутов, Список операций).

Имя класса – уникальный идентификатор класса, отличающий его от других.

Каждый атрибут тоже представляется некоторым именем и может принимать значения определенного типа. По существу, атрибут класса выражает некоторое простое свойство объектов этого класса. Представление некоторых простых свойств объектов атрибутами весьма удобно, особенно когда по значениям этих атрибутов осуществляется классификация объектов. (если число ног у некоего существа равно шести, то это существо относится к классу насекомых).

Операции, указываемые в представлении класса, отражают другие свойства объектов этого класса (как простые, так и ассоциативные). Они показывают, что можно делать с объектами этого класса (или что могут делать сами эти объекты).

В объектной модели *отношение между объектами* некоторых классов обобщаются в

отношения между этими *классами*. При этом используются, как правило, только одноместные и двуместные отношения между объектами. Более сложные отношения приводит к неоправданному усложнению объектных моделей, а с другой стороны, такие отношения всегда могут быть сведены к двуместным за счет определения дополнительных классов. Одноместные отношения называют *атрибутами*, причем некоторые атрибуты объекта получаются из атрибутов класса присвоением им конкретных значений. Отношение между двумя (и более) объектами называют *связями*, а их обобщение (отношение между классами) обычно называют *ассоциациями*. Ассоциации определяют допустимые связи между объектами. Различают следующие виды ассоциаций:

- взаимодействия состояний объектов,
- агрегирования (структурирования) объектов,
- абстрагирования (порождения) классов.

Ассоциация «взаимодействие», по существу, означает, что объекты классов, находящихся в таком отношении, могут быть параметрами некоторых операций.

Ассоциация «агрегирование» означает, что объект одного из классов, находящихся в таком отношении, включает (или может включать) в себя (как часть) объекты другого из этих классов.

Ассоциация «абстрагирование» означает, что один из классов, находящихся в таком отношении, наследует свойства другого из этих классов и может обладать также и другими (дополнительными) свойствами.

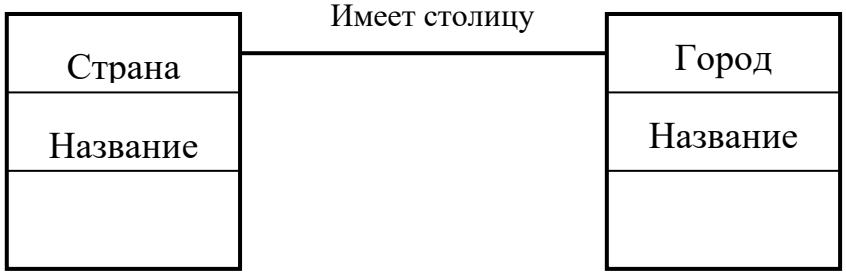

Рис.2. Пример отношения между классами объектов.

Для представления объектной модели часто используются графические языки спецификации объектов (например, язык UML).

На таких языках классы и объекты задаются прямоугольниками, в которых указывается специфицирующая их информация. Для задания отношений между двумя классами соответствующие им прямоугольники связываются линией, снабженной различными графическими значками и некоторыми надписями. Графические значки специфицируют характер (вид) отношения между этими классами, а надписи обеспечивают полную идентификацию этого отношения (делают его конкретным). Например, на рис.2 заданное отношение между классами Страна и Город имеет характер «один к одному». Более конкретно это отношение означает, что каждый объект класса Страна обязательно связан отношением «имеет столицей» с одним и только одним объектом класса Город, и этот объект класса Город не связан таким отношением ни с каким другим объектом класса Страна.

Для описания декомпозиции объектов используется отношение вида агрегирования. Например, отношение «программа состоит из одного или нескольких модулей» представлено на рис.3.

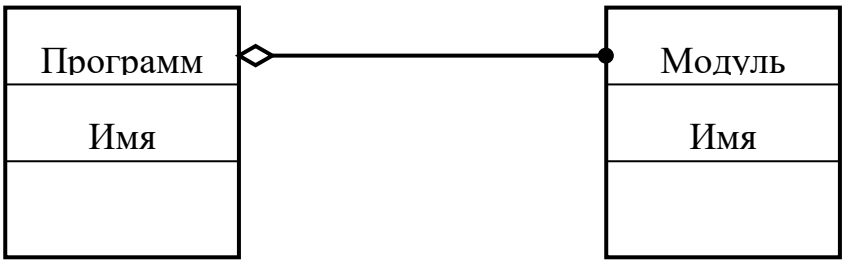

Рис. 3. Пример отношения агрегирования между классами объектов.

Следует заметить, что объектная модель полностью включает описание внешней информационной среды при процедурном подходе.

Динамическая модель показывает допустимые последовательности изменений состояний объектов из объектной модели модельного мира, который должно представлять разрабатываемое ПС (программная система). Она описывает последовательности операций в ответ на внешние сигналы (взаимодействия) без рассмотрения того, что эти операции делают. Динамическая модель необходима, если в соответствующей объектной модели имеются активные объекты.

Основные понятия динамической модели: события и состояния объектов. Под *событием* здесь понимается элементарное воздействие одного объекта на другой, происходящее в определенный момент времени. Одно событие может логически предшествовать другому или быть не связанным с другим. Другими словами, события в динамической модели *частично упорядочены*. Под *состоянием объекта* здесь понимается совокупность значений атрибутов объекта и представления текущих связей этого объекта с другими объектами. Состояние объекта связывается с интервалом времени между некоторыми двумя событиями, на которые реагирует этот объект. Объект *переходит* из одного состояния в другое в результате реакции на некоторое событие (в конце интервала, связанного с этим состоянием).

В связи с этим в динамической модели для каждого класса активных объектов строится своя *диаграмма состояний*. Она представляет собой граф, вершинами которого являются состояния, а дугами – переходы между этими состояниями (переходы часто обозначаются именами событий). Некоторые переходы могут быть связаны с условиями, разрешающими эти переходы. *Условие* – это предикат, зависящий от значений некоторых атрибутов объекта. Каждое условие указывается на дуге, переходом по которой управляет это условие. Существенно, что в диаграмме состояний с некоторыми состояниями или событиями связываются определенные операции.

Операция, связываемая с событием, обозначает реакцию объекта на это событие и считается, что она выполняется мгновенно (в точке некоторого временного интервала). Такая операция называется *действием*.

Операция, связываемая с состоянием, выполняется в рамках временного интервала, с которым связано это состояние (т.е. имеет продолжительность, ограниченную этим интервалом). Такая операция называется *деятельностью*.

Диаграмма состояний определяет управление активизацией указанных операций. Таким образом, диаграмма состояний описывает поведение одного класса объектов. Пример диаграммы состояний класса приведен на рис.4.

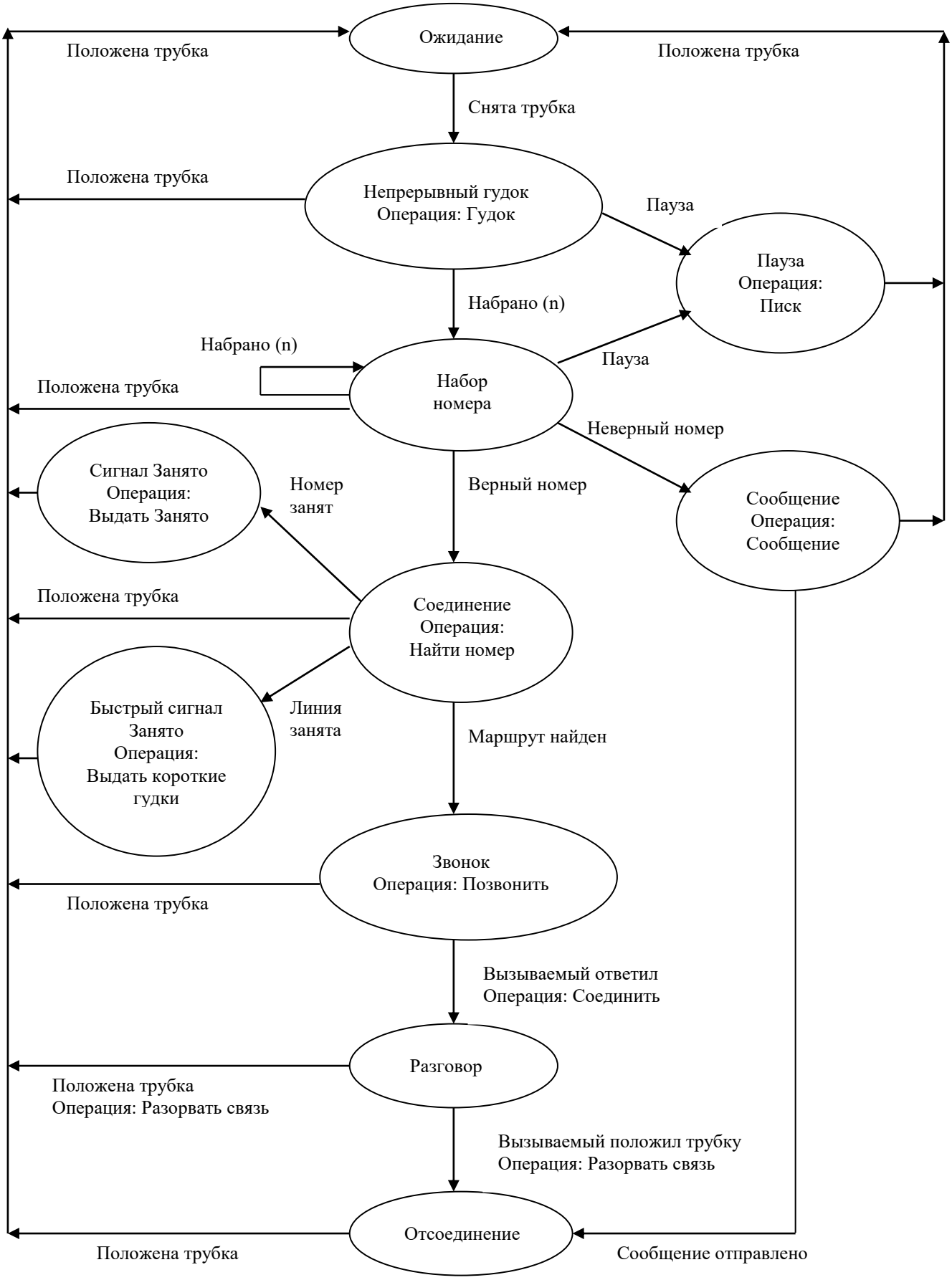

Рис.4. Диаграмма состояний телефонной линии.

Динамическая модель в целом объединяет все диаграммы состояний с помощью событий между классами.

Функциональная модель показывает, как вычисляются выходные значения из входных без указания порядка, в котором эти значения вычисляются. Она определяет все операции, условия и

ограничения, используемые в объектной и динамической моделях (*внешние операции*). Функциональная модель соответствует определению *внешних функций* при процедурном подходе к разработке ПС.

Для определения крупных операций в функциональной модели используются *потоковые диаграммы* (*диаграммы потоков данных*), позволяющие выразить эти операции через более простые операции.

Основными понятиями потоковых диаграмм являются *процессы*, *объекты* и *потоки данных*. Потоковая диаграмма – это граф, вершинами которого являются объекты или процессы, а дугами – потоки данных. Процессы преобразуют данные, поступающие от одних объектов и направляемые для хранения в другие объекты. Эти процессы представляют *внутренние* операции, через которые выражается операция, представляемая данной потоковой диаграммой. Объекты могут быть пассивными (*хранилищами данных*) и активными (*агентами*). Пассивные объекты используются только для хранения данных, а активные объекты используются как для хранения, так и для преобразования данных. Потоки данных определяют допустимые направления перемещения данных и типы перемещаемых данных.

Процессы могут выражаться терминальными операциями (определяемые непосредственно) или с помощью других потоковых диаграмм. Таким образом, потоковые диаграммы являются иерархическими.

Таким образом, основным содержанием этапа внешнего описания при объектном подходе является *объектное моделирование*. При этом широко используются формальные языки спецификаций, в том числе и графические. Одним из наиболее употребительных в настоящее время таких языков является язык UML.

2.3. Особенности объектного подхода на этапе конструирования программного средства.

На этапе конструирования при объектном подходе продолжается процесс объектного моделирования: уточняются модели, построенные на этапе внешнего описания, в терминах описания программных систем и производится дальнейшая декомпозиция объектов.

В процессе разработки объектной архитектуры ПС выделяются все объекты, с информационными моделями которых собирается непосредственно работать пользователь, и завершается их программная спецификация, а так же определяется их пользовательский интерфейс. Такие объекты мы будем называть *пользовательскими*. Классы таких объектов или отдельные активные объекты образуют архитектурные подсистемы. Определяется метод взаимодействия между этими подсистемами.

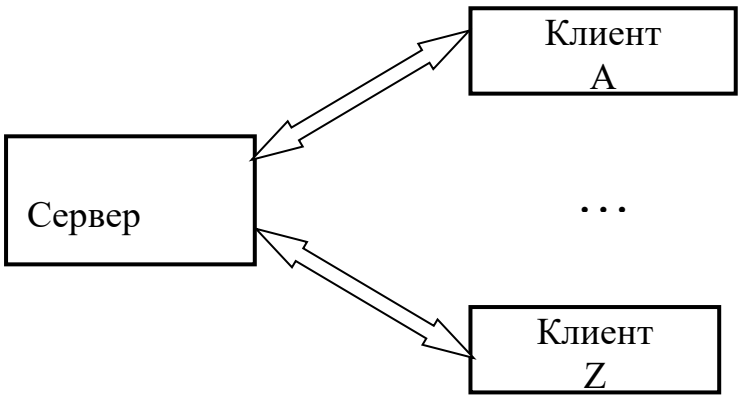

Рис.5. Архитектура «Клиент-Сервер»

В случае использования активных объектов основным широким классом архитектур при объектном подходе является *коллектив параллельно действующих программ*, причем здесь роль программ выполняют как раз эти активные объекты, а способ управления передачей сообщений зависит от выбранного подкласса таких архитектур. Типичной архитектурой такого класса является архитектура «клиент-сервер» (см. рис.5). В такой системе один из активных объектов, называемый *сервером*, выполняет определенные программные услуги по запросам других активных объектов, называемых *клиентами*. Такой запрос передается серверу с помощью сообщения от клиента, результат выполнения сервером запроса передаются соответствующему

клиенту с помощью другого сообщения.

Дальнейшая разработка структуры программных подсистем и их кодирование на языках программирования может осуществляться уже в рамках процедурного подхода на ориентированных на него языках программирования – пользователь внутреннюю организацию этих подсистем уже «не видит». Однако во многих случаях существуют сильные аргументы за то, чтобы продолжить объектную декомпозицию этих подсистем. Объектная структура этих подсистем может быть существенно более понятной *разработчику*, чем их структура при процедурном подходе. Кроме того, продолжение объектной декомпозиции и использование основных понятий и методов объектного подхода при дальнейшей разработке ПС представляется «естественным», так как весь процесс разработки становится единообразным (концептуально целостным).

При разработки модульной структуры при объектном подходе используются преимущественно информационно прочные модули. Информационно прочный модуль, реализующий какой-либо класс объектов или логически связанную совокупность таких классов, обычно называют *компонентом* ПС.

2.4. Особенности объектного подхода на этапе кодирования программного средства.

На этапе кодирования при объектном подходе используются языки программирования уже другого типа – *объектно-ориентированные*. Считается, что язык программирования поддерживает объектно-ориентированное программирование, если он включает конструкции для:

- инкапсуляции и абстракции данных,
- наследования,
- динамического полиморфизма.

Однако самое существенное различие между процедурными и объектно - ориентированными языками программирования заключается в следующем: в качестве основных средств декомпозиции программ в процедурных языках используются описания функций и процедур (декомпозиция отношений), а в объектно - ориентированных языках – описание классов объектов (декомпозиция объектов). Объекты, возникающие в программах при такой декомпозиции архитектурных подсистем, мы будем называть *объектами процесса выполнения программ*.

Изменяется и технология разработки программного модуля. При процедурном подходе для разработки модуля (в основном, реализующего функцию или процедуру) рекомендовались структурное программирование и пошаговая детализация. При объектно - ориентированном подходе для разработки компонента (информационно прочного модуля) более подходит т.н. нисходящая технологию разработки *слоистой* структуры модуля. В этой технологии каждый модуль рассматривается состоящим из некоторой упорядоченной совокупности программных слоев, причем каждый слой является реализацией некоторой абстрактной машины (в нашем случае это можно рассматривать как реализацию класса или логически связанной совокупности классов). Таким образом, модуль образует определенная цепочка иерархически связанных абстрактных машин (так называемые «бусы») Реализация каждой абстрактной машины выражается в терминах ниже стоящей абстрактной машины (ниже стоящего программного слоя) или в терминах выбранного языка программирования (при реализации абстрактной машины самого нижнего уровня). Разработка модуля начинается с разработки абстрактной машины самого верхнего уровня.

Прототипом объектно-ориентированного программирования послужил ряд средств, входящих в состав языка SIMULA-67. Но в самостоятельный стиль оно оформилось с появлением языка SMALLTALK, разработанного А. Кеем в 1972 году и первоначально предназначенного для реализации функций машинной графики.

В основе объектно-ориентированного стиля программирования лежит понятие объекта, а суть его выражается формулой: «объект = данные + процедуры». Каждый объект интегрирует в себе некоторую структуру данных и доступные только ему процедуры обработки этих данных, называемые методами. Объединение данных и процедур в одном объекте называется инкапсуляцией и присуще объектно-ориентированному программированию.

Для описания объектов служат классы. Класс определяет свойства и методы объекта,

принадлежащего этому классу. Соответственно, любой объект можно определить как экземпляр класса.

Программирование рассматриваемого стиля заключается в выборе имеющихся или создании новых объектов и организации взаимодействия между ними. При создании новых объектов свойства объектов могут добавляться или наследоваться от объектов-предков. В процессе работы с объектами допускается полиморфизм — возможность использования методов с одинаковыми именами для обработки данных разных типов.

К современным наиболее объектно-ориентированным языкам программирования относятся C++ и Java.

Язык C++ был разработан в начале 80-х годов Б. Страуструпом, сотрудником лаборатории Bell корпорации AT&T. Им была создана компактная компилирующая система, в которой за основу был взят язык C, дополненный элементами языков BCPL, Simula-67 и Algol - 68. К июлю 1983 года появился язык C с классами, а чуть позднее — C++. К 1990 году была выпущена третья версия языка C++, принятая комитетом ANSI в качестве исходного материала для его стандартизации.

В 1990 году сотрудник корпорации Sun Д. Гослинг на основе расширения C++ разработал объектно-ориентированный язык Oak, основным достоинством которого было обеспечение сетевого взаимодействия различных по типу устройств. Новая интегрируемая в Internet версия языка, получила название Java. Первый броузер, который поддерживал язык Java, разработан программистом корпорации Sun П. Нафтоном и получил название HotJava. С января 1995 года Java получает распространение в Internet.

Согласно официальному определению авторов, Java является простым объектно - ориентированным и архитектурно-нейтральным языком интерпретирующего типа, обеспечивающим надежность, безопасность и переносимость, обладающим высокой производительностью в сочетании с многопоточностью и динамичностью.

Принципиальной разницей между Java и C++ является то, что первый из них является интерпретируемым, а второй — компилируемым. Синтаксис языков практически полностью совпадает.

С точки зрения возможностей собственно объектно-ориентированных средств язык Java обладает рядом преимуществ перед языком C++. Так, язык Java демонстрирует более гибкую и мощную систему инкапсуляции информации. Механизм наследования, реализованный в Java, обязывает к более строгому подходу к программированию, что улучшает надежность и понимаемость кода. Язык же C++ обладает сложной, неадекватной и трудной для понимания системой наследования. Возможности динамического связывания объектов одинаково хорошо представлены в обоих языках, однако, синтаксическая избыточность C++ заставляет и здесь отдать предпочтение языку Java.

В силу своей конструктивности идеи объектно-ориентированного программирования используются во многих универсальных процедурных языках. Так, например, в состав интегрированной системы программирования на языке PASCAL (корпорации Borland International) версии 5.5 входит специальная библиотека объектно - ориентированного программирования Turbo Vision.

В последнее время многие программы, в особенности объектно-ориентированные, реализуются как системы визуального программирования. Отличительной особенностью таких систем является мощная среда разработки программ из готовых «строительных блоков», позволяющая создать интерфейсную часть программного продукта в диалоговом режиме, практически без кодирования программных операций. К числу объектно-ориентированных систем визуального программирования относятся: Visual Basic, Delphi, C++ Builder и Visual C++.

Контрольные вопросы:

1. В чем заключается сущность процедурного программирования?
2. Какие языки процедурного программирования вам известны? Для чего они предназначены?
3. В чем заключается сущность функционального программирования?
4. Чем характеризуется логическое программирование?
5. Опишите принцип объектно-ориентированного подхода к программированию.
6. В чем заключается особенность объектного подхода к разработке внешнего описания программного средства?
7. В чем заключается особенность объектного подхода на этапе конструирования ПС?
8. В чем заключается особенность объектного подхода на этапе кодирования ПС?