**41 Обработка исключительных ситуаций. Реализация на уровне объектов.**

**Общее понятие исключительной ситуации**

Во время выполнения программы могут возникать ситуации, когда состояние данных, устройств ввода-вывода или компьютерной системы в целом делает дальнейшие вычисления в соответствии с базовым алгоритмом невозможными или бессмысленными. Классические примеры подобных ситуаций приведены ниже.

* Целочисленное деление на ноль. Результата у операции быть не может, поэтому ни дальнейшие вычисления, ни попытка использования результата деления не приведут к решению задачи.
* Ошибка при попытке считать данные с внешнего устройства. Если данные не удаётся ввести, любые дальнейшие запланированные операции с ними бессмысленны.
* Исчерпание доступной памяти. Если в какой-то момент система оказывается не в состоянии выделить достаточный для прикладной программы объём оперативной памяти, программа не сможет работать нормально.
* Появление сигнала аварийного отключения электропитания системы. Прикладную задачу, по всей видимости, решить не удастся, в лучшем случае (при наличии какого-то резерва питания) прикладная программа может позаботиться о сохранении данных.
* Появление на входе коммуникационного канала данных, требующих немедленного считывания. Чем бы ни занималась в этот момент программа, она должна перейти к чтению данных, чтобы не потерять поступившую информацию.

**Виды исключительных ситуаций**

Исключительные ситуации, возникающие при работе программы, можно разделить на два основных типа: синхронные и асинхронные, принципы реакции на которые существенно различаются.

* **Синхронные исключения** могут возникнуть только в определённых, заранее известных точках программы. Так, ошибка чтения файла или коммуникационного канала, нехватка памяти — типичные синхронные исключения, так как возникают они только в операции чтения из файла или из канала или в операции выделения памяти соответственно.
* **Асинхронные исключения** могут возникать в любой момент времени и не зависят от того, какую конкретно инструкцию программы выполняет система. Типичные примеры таких исключений: аварийный отказ питания или поступление новых данных.

Некоторые типы исключений могут быть отнесены как к синхронным, так и к асинхронным. Например, инструкция деления на нуль формально должна приводить к синхронному исключению, так как логически оно возникает именно при выполнении данной команды, но на некоторых платформах за счёт глубокой конвейеризации исключение может фактически оказаться асинхронным.

## Обработчики исключений

### Общее описание

В отсутствие собственного механизма обработки исключений для прикладных программ наиболее общей реакцией на любую исключительную ситуацию является немедленное прекращение выполнения с выдачей пользователю сообщения о характере исключения. Можно сказать, что в подобных случаях единственным и универсальным обработчиком исключений становится операционная система. Например, в операционную систему [Windows](http://dic.academic.ru/dic.nsf/ruwiki/754819) встроена утилита [Dr. Watson](http://dic.academic.ru/dic.nsf/ruwiki/1267722), которая занимается сбором информации о необработанном исключении и её отправкой на специальный сервер компании [Microsoft](http://dic.academic.ru/dic.nsf/ruwiki/7585).

Возможно игнорирование исключительной ситуации и продолжение работы, но такая тактика опасна, так как приводит к ошибочным результатам работы программ или возникновению ошибок впоследствии. Например, проигнорировав ошибку чтения из файла блока данных, программа получит в своё распоряжение не те данные, которые она должна была считать, а какие-то другие. Результаты их использования предугадать невозможно.

Обработка исключительных ситуаций самой программой заключается в том, что при возникновении исключительной ситуации управление передаётся некоторому заранее определённому *обработчику* — блоку кода, процедуре, функции, которые выполняют необходимые действия.

Существует два принципиально разных механизма функционирования обработчиков исключений.

* **Обработка с возвратом** подразумевает, что обработчик исключения ликвидирует возникшую проблему и приводит программу в состояние, когда она может работать дальше по основному алгоритму. В этом случае после того, как выполнится код обработчика, управление передаётся обратно в ту точку программы, где возникла исключительная ситуация (либо на команду, вызвавшую исключение, либо на следующую за ней, как в некоторых старых диалектах языка BASIC) и выполнение программы продолжается. Обработка с возвратом типична для обработчиков асинхронных исключений (которые обычно возникают по причинам, не связанным прямо с выполняемым кодом), для обработки синхронных исключений она малопригодна.
* **Обработка без возврата** заключается в том, что после выполнения кода обработчика исключения управление передаётся в некоторое, заранее заданное место программы, и с него продолжается исполнение. То есть, фактически, при возникновении исключения команда, во время работы которой оно возникло, заменяется на [безусловный переход](http://dic.academic.ru/dic.nsf/ruwiki/803812) к заданному оператору.

Существует два варианта подключения обработчика исключительных ситуаций к программе: структурная и неструктурная обработка исключений.

### Неструктурная обработка исключений

Неструктурная обработка исключений реализуется в виде механизма регистрации [функций](http://dic.academic.ru/dic.nsf/ruwiki/7543) или команд-обработчиков для каждого возможного [типа](http://dic.academic.ru/dic.nsf/ruwiki/4898) исключения. [Язык программирования](http://dic.academic.ru/dic.nsf/ruwiki/1465) или его системные [библиотеки](http://dic.academic.ru/dic.nsf/ruwiki/10141) предоставляют программисту как минимум две стандартные процедуры: регистрации обработчика и разрегистрации обработчика. Вызов первой из них «привязывает» обработчик к определённому исключению, вызов второй — отменяет эту «привязку». Если исключение происходит, выполнение основного кода программы немедленно прерывается и начинается выполнение обработчика. По завершении обработчика управление передаётся либо в некоторую наперёд заданную точку программы, либо обратно в точку возникновения исключения (в зависимости от заданного способа обработки — с возвратом или без). Независимо от того, какая часть программы в данный момент выполняется, на определённое исключение всегда реагирует последний зарегистрированный для него обработчик. В некоторых языках зарегистрированный обработчик сохраняет силу только в пределах текущего блока кода (процедуры, функции), тогда процедура разрегистрации не требуется. Ниже показан условный фрагмент кода программы с неструктурной обработкой исключений:

УстановитьОбработчик(ОшибкаБД, ПерейтиНаОшБД)

// На исключение "ОшибкаБД" установлен обработчик - команда "ПерейтиНаОшБД"

... // Здесь находятся операторы работы с БД

ПерейтиНаСнятьОшБД // Команда безусловного перехода - обход обработчика исключений

ОшБД: // метка - сюда произойдёт переход в случае ошибки БД по установленному обработчику

... // Обработчик исключения БД

СнятьОшБД:

// метка - сюда произойдёт переход, если контролируемый код выполнится без ошибки БД.

СнятьОбработчик(ОшибкаБД)

// Обработчик снят

Неструктурная обработка — практически единственный вариант для обработки асинхронных исключений, но для синхронных исключений она неудобна: приходится часто вызывать команды установки/снятия обработчиков, всегда остаётся опасность нарушить логику работы программы, пропустив регистрацию или разрегистрацию обработчика.

### Структурная обработка исключений

Структурная обработка исключений требует обязательной поддержки со стороны языка программирования — наличия специальных синтаксических конструкций. Такая конструкция содержит блок контролируемого кода и обработчик (обработчики) исключений. Наиболее общий вид такой конструкции (условный):

НачалоБлока

... // Контролируемый код

...

если (условие) то СоздатьИсключение Исключение2

...

Обработчик Исключение1

... // Код обработчика для Исключения1

Обработчик Исключение2

... // Код обработчика для Исключения2

ОбработчикНеобработанных

... // Код обработки ранее не обработанных исключений

КонецБлока

Здесь «НачалоБлока» и «КонецБлока» — ключевые слова, которые ограничивают блок контролируемого кода, а «Обработчик» — начало блока обработки соответствующего исключения. Если внутри блока, от начала до первого обработчика, произойдёт исключение, то произойдёт переход на обработчик, написанный для него, после чего весь блок завершится и исполнение будет продолжено со следующей за ним команды. В некоторых языках нет специальных ключевых слов для ограничения блока контролируемого кода, вместо этого обработчик (обработчики) исключений могут быть встроены в некоторые или все синтаксические конструкции, объединяющие несколько операторов. Так, например, в языке Ада любой составной оператор (begin — end) может содержать обработчик исключений.

«ОбработчикНеобработанных» — это обработчик исключений, которые не соответствуют ни одному из описанных выше в данном блоке. Обработчики исключений в реальности могут описываться по-разному (один обработчик на все исключения, по одному обработчику на каждый тип исключение), но принципиально они работают одинаково: при возникновении исключения находится первый соответствующий ему обработчик в данном блоке, его код выполняется, после чего выполнение блока завершается. Исключения могут возникать как в результате программных ошибок, так и путём явной их генерации с помощью соответствующей команды (в примере — команда «СоздатьИсключение»). С точки зрения обработчиков такие искусственно созданные исключения ничем не отличаются от любых других.

Блоки обработки исключений могут многократно входить друг в друга, как явно (текстуально), так и неявно (например, в блоке вызывается процедура, которая сама имеет блок обработки исключений). Если ни один из обработчиков в текущем блоке не может обработать исключение, то выполнение данного блока немедленно завершается, и управление передаётся наближайший подходящий обработчик более высокого уровня иерархии. Это продолжается до тех пор, пока обработчик не найдётся и не обработает исключение или пока не выйдет из обработчиков заданных программистом и не будет передано системному обработчику по умолчанию, аварийно закроющему программу.

Иногда бывает неудобно завершать обработку исключения в текущем блоке, то есть желательно, чтобы при возникновении исключения в текущем блоке обработчик выполнил какие-то действия, но исключение продолжило бы обрабатываться на более высоком уровне (обычно так бывает, когда обработчик данного блока не полностью обрабатывает исключение, а лишь частично). В таких случаях в обработчике исключений генерируется новое исключение или возобновляется с помощью специальной команды ранее появившееся. Код обработчиков не является защищённым в данном блоке, поэтому созданное в нём исключение будет обрабатываться в блоках более высокого уровня.

### Блоки с гарантированным завершением

Помимо блоков контролируемого кода для обработки исключений, языки программирования могут поддерживать блоки с гарантированным завершением. Их использование оказывается удобным тогда, когда в некотором блоке кода, независимо от того, произошли ли какие-то ошибки, необходимо перед его завершением выполнить определённые действия. Простейший пример: если в процедуре динамически создаётся какой-то локальный объект в памяти, то перед выходом из этой процедуры объект должен быть уничтожен (чтобы избежать утечки памяти), независимо от того, произошли после его создания ошибки или нет. Такая возможность реализуется блоками кода вида:

НачалоБлока

... // Основной код

Завершение

... // Код завершения

КонецБлока

Заключённые между ключевыми словами «НачалоБлока» и «Завершение» операторы (основной код) выполняются последовательно. Если при выполнении их не возникает исключений, то затем выполняются операторы между ключевыми словами «Завершение» и «КонецБлока» (код завершения). Если же при выполнении основного кода возникает исключение (любое), то сразу же выполняется код завершения, после чего весь блок завершается, а возникшее исключение продолжает существовать и распространяться до тех пор, пока его не перехватит какой-либо блок обработки исключений более высокого уровня.

Принципиальное отличие блока с гарантированным завершением от обработки — то, что он не обрабатывает исключение, а лишь гарантирует выполнение определённого набора операций перед тем, как включится механизм обработки. Стоит заметить, что блок с гарантированным завершением легко реализуется с помощью команд «возбудить исключение» и «структурный обработчик исключения».

**42 Стадии разработки ПО.**

В процессе создания любой программы можно выделить несколько этапов.

1. *Постановка задачи —* выполняется специалистом в предметной области на естественном языке (русском, английском и т. д.). Необходимо определить цель задачи, ее содержание и общий подход к решению. Возможно, что задача решается точно*(аналитически),* и без компьютера можно обойтись. Уже на этапе постановки надо учитывать эффективность алгоритма решения задачи на ЭВМ, ограничения, накладываемые *аппаратным и программным обеспечением* (АО и ПО).

2. *Анализ задачи и моделирование —* определяются исходные данные и результат, выявляются ограничения на их значения, выполняется формализованное описание задачи и построение (выбор) математической модели, пригодной для решения на компьютере.

3. *Разработка или выбор алгоритма решения задачи —* выполняется на осно­ве ее математического описания. Многие задачи можно решить различными способами. Программист должен выбрать оптимальное решение. Неточности в постановке, анализе задачи или разработке алгоритма могут привести к *скрытой ошибке —* программист получит неверный результат, считая его правильным.

4. *Проектирование общей структуры программы —* формируется модель решения с последующей детализацией и разбивкой на подпрограммы, определяется "архитектура" программы, способ хранения информации (набор переменных, массивов и т. п.).

5. *Кодирование —* запись алгоритма на языке программирования. Современные системы программирования позволяют ускорить процесс разработки программы, автоматически создавая часть ее текста, однако творческая работа по-прежнему лежит на программисте. Для успешной реализации целей проекта программисту необходимо использовать *методы структурного программирования.*

6. *Отладка и тестирование программы.* Под *отладкой* понимается устранение ошибок в программе. *Тестирование* позволяет вести их поиск и, в конечном счете, убедиться в том, что полностью отлаженная программа дает правильный результат. Для этого разрабатывается *система тестов —*специально подобранных контрольных примеров с такими наборами па­раметров, для которых решение задачи известно. Тестирование должно охватывать все возможные ветвления в программе, т. е. *проверять все ее инструкции,*и включать такие исходные данные, для которых решение *невозможно.* Проверка особых, *исключительных ситуаций,* необходима для анализа корректности. Например, программа должна отказать клиенту банка в просьбе выдать сумму, отсутствующую на его счете. В ответст­венных проектах большое внимание уделяется так называемой "защите от дурака" подразумевающей устойчивость программы к неумелому обращению пользователя. Использование специальных программ — *отладчиков,* которые позволяют выполнять программу по отдельным шагам, просматривая при этом значения переменных, значительно упрощает этот этап.

7. *Анализ результатов —* если программа выполняет моделирование какого-либо известного процесса, следует сопоставить результаты вычислений с результатами наблюдений. В случае существенного расхождения необ­ходимо изменить модель.

8. *Публикация* результатов работы, *передача заказчику* для эксплуатации.

9. *Сопровождение программы —* включает консультации представителей заказчика по работе с программой и обучение персонала. Недостатки и ошибки, замеченные в процессе эксплуатации, должны устраняться.

**42. Проектирование модели состояний программной системы.**

Объектная модель представляет статическую структуру проектируемой системы (подсистемы). Однако знания статической структуры недостаточно, чтобы понять и оценить работу подсистемы. Необходимо иметь средства для описания изменений, которые происходят с объектами и их связями во время работы подсистемы. Одним из таких средств является динамическая модель подсистемы. Она строится после того, как объектная модель подсистемы построена и предварительно согласована и отлажена. Динамическая модель подсистемы состоит из диаграмм состояний ее объектов и подсистем.

Текущее состояние объекта характеризуется совокупностью текущих значений его атрибутов и связей. Во время работы системы составляющие ее объекты взаимодействуют друг с другом, в результате чего изменяются их состояния. Единицей влияния является событие: каждое событие приводит к смене состояния одного или нескольких объектов в системе, либо к возникновению новых событий. Работа системы характеризуется последовательностью происходящих в ней событий.

*События*

Событие происходит в некоторый момент времени (нередко оно используется для определения соответствующего момента времени). Примеры событий: старт ракеты, старт забега на 100 м, начало проводки (в банковской сети), выдача денег и т.п. Событие не имеет продолжительности (точнее, оно занимает пренебрежимо малое время).

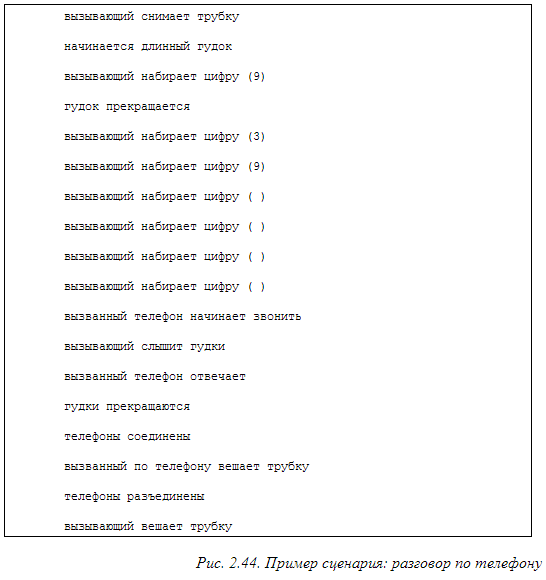
Одно из событий может логически предшествовать другому, либо следовать за другим, либо они могут быть независимыми; примерами логически (причинно) связанных событий являются старт и финиш одного забега, начало проводки и выдача денег клиенту (в результате этой проводки), примерами независимых событий - старт ракеты и финиш забега, проводки, запущенные с разных ATM, и т.п. Если события не имеют причинной связи (т.е. они логически независимы), они называются *независимыми*(concurrent); такие события не влияют друг на друга. Независимые события не имеет смысла упорядочивать, так как они могут происходить в произвольном порядке. Модель распределенной системы обязательно должна содержать независимые события и активности.

События передают информацию с одного объекта на другой. Существуют классы событий, которые просто сигнализируют о том, что что-то произошло или происходит (примеры: загорание лампочки лифта, гудок в телефонной трубке). В программировании рассматриваются *исключительные события*(иногда их называют исключениями), которые сигнализируют о нарушениях работы аппаратуры, либо программного обеспечения.

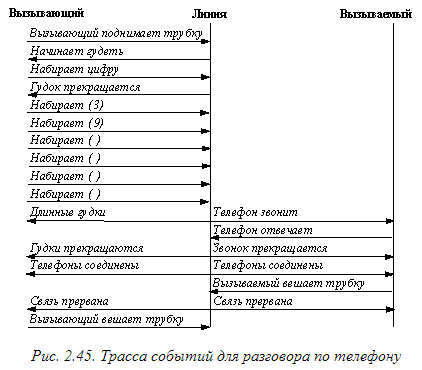
*Сценарии и трассы событий*

Сценарием называется последовательность событий, которая может иметь место при конкретном выполнении системы. Сценарии могут иметь разные области влияния: они могут включать все события, происходящие в системе, либо только события, возникающие и влияющие только на определенные объекты системы.

На рисунке 2.44 приведен пример сценария пользования телефонной линией. Каждое событие в этом сценарии передает информацию с одного объекта на другой; например событие начинается длинный гудок передает сигнал от телефонной линии к вызывающему (пользователю). При анализе динамики работы системы необходимо составить и рассмотреть несколько сценариев, отражающих типичные варианты ее работы.



Следующим этапом после разработки и анализа сценариев является определение объектов, генерирующих и принимающих каждое событие сценария. Последовательности событий с привязкой к объектам проектируемой системы удобно представлять на диаграммах, называемых трассами событий. Пример трассы событий для разговора по телефону представлен на рисунке 2.45. Вертикальные линии изображают на этой диаграмме объекты, а горизонтальные стрелки - события (стрелка начинается в объекте, генерирующем событие, и заканчивается в объекте, принимающем событие). Более поздние события помещены ниже более ранних, одновременные - на одном уровне.

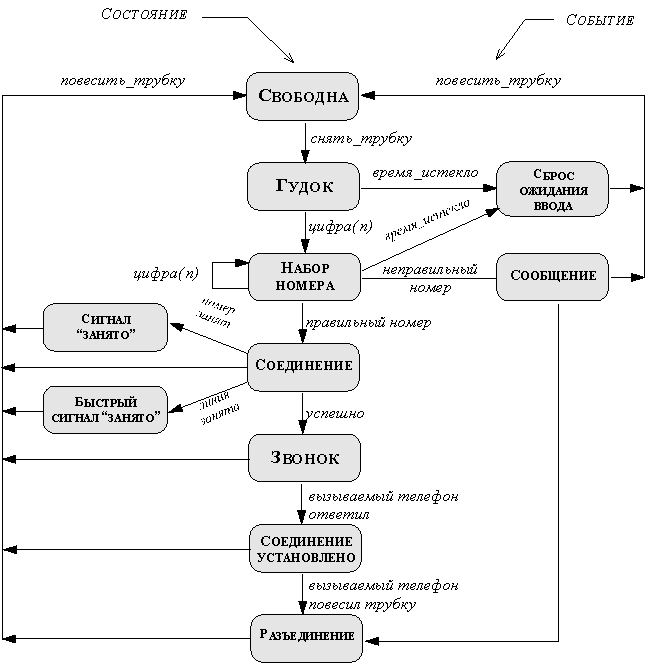


*Состояния*

Состояние определяется совокупностью текущих значений атрибутов и связей объекта. Например, банк может иметь состояния платежеспособный и неплатежеспособный (когда большая часть банков одновременно оказывается во втором состоянии, наступает банковский кризис).Состояние определяет реакцию объекта на поступающее в него событие (в том, что реакция различна нетрудно убедиться с помощью банковской карточки: в зависимости от состояния банка обслуживание (реакция банка на предъявление карточки) будет разным). Реакция объекта на событие может включать некоторое действие и/или перевод объекта в новое состояние.Объект сохраняет свое состояние в течение времени между двумя последовательными событиями, которые он принимает: события представляют моменты времени, состояния - отрезки времени; состояние имеет продолжительность, которая обычно равна отрезку времени между двумя последовательными событиями, принимаемыми объектом, но иногда может быть больше.При определении состояний мы не рассматриваем тех атрибутов, которые не влияют на поведение объекта, и объединяем в одно состояние все комбинации значений атрибутов и связей, которые дают одинаковые реакции на события.

*Диаграммы состояний*

Диаграмма состояний связывает события и состояния. При приеме события следующее состояние системы зависит как от ее текущего состояния, так и от события; смена состояния называется *переходом*. Диаграмма состояний - это граф, узлы которого представляют состояния, а направленные деги, помеченные именами соответствующих событий, - переходы. Диаграмма состояний позволяет получить последовательность состояний по заданной последовательности событий.



*Рис. 2.46. Диаграмма состояний телефонной линии*

**43 Проектирование функциональной модели программной системы.**

Функциональная модель показывает как вычисляются значения в системе и как они зависят одно от другого. Для конструирования функциональной модели необходимо выполнить следующее:

* определить входные и выходные значения;
* построить ДПД, показывающие функциональные зависимости;
* описать функции;
* описать ограничения;
* сформулировать критерии оптимизации;
* уточнить набор операций в объектной модели.

Функциональная модель описывает вычисления в системе. Она показывает, каким образом выходные данные вычисляются по входным данным, не рассматривая порядок и способ реализации вычислений. Функциональная модель состоит из набора диаграмм потока данных, которые показывают потоки значений от внешних входов через операции и внутренние хранилища данных к внешним выходам. Функциональная модель описывает смысл операций объектной модели и действий динамической модели, а также ограничения на объектную модель. Неинтерактивные программы (например, компиляторы) имеют тривиальную динамическую модель: их цель состоит в вычислении значения некоторой функции. Основной моделью таких программ является функциональная модель (хотя если программа имеет нетривиальные структуры данных, для нее важна и объектная модель).

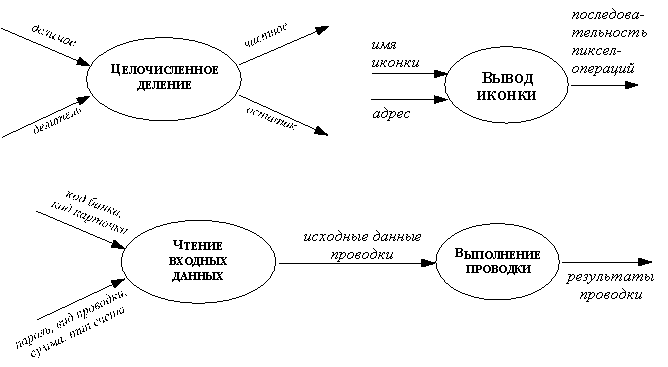
Функциональная модель представляет собой набор диаграмм потоков данных (далее - ДПД), которые описывают смысл операций и ограничений. ДПД отражает функциональные зависимости значений, вычисляемых в системе, включая входные значения, выходные значения и внутренние хранилища данных. ДПД - это граф, на котором показано движение значений данных от их источников через преобразующие их процессы к их потребителям в других объектах.

ДПД содержит процессы, которые преобразуют данные, потоки данных, которые переносят данные, активные объекты, которые производят и потребляют данные, и хранилища данных, которые пассивно хранят данные.

*Процессы*

Процесс преобразует значения данных. Процессы самого нижнего уровня представляют собой функции без побочных эффектов (примерами таких функций являются вычисление суммы двух чисел, вычисление комиссионного сбора за выполнение проводки с помощью банковской карточки и т.п.). Весь граф потока данных тоже представляет собой процесс (высокого уровня). Процесс может иметь побочные эффекты, если он содержит нефункциональные компоненты, такие как хранилища данных или внешние объекты.

На ДПД процесс изображается в виде эллипса, внутри которого помещается имя процесса; каждый процесс имеет фиксированное число входных и выходных данных, изображаемых стрелками (см. рисунок 2.60).

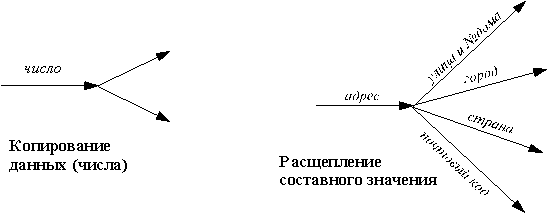


*Рис. 2.60. Примеры процессов*

Процессы реализуются в виде методов (или их частей) и соответствуют операциям конкретных классов.

*Потоки данных*

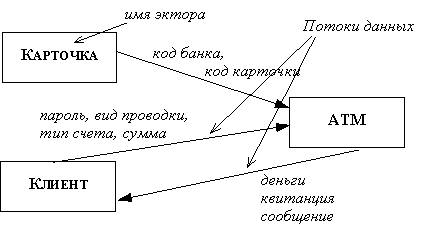
Поток данных соединяет выход объекта (или процесса) со входом другого объекта (или процесса). Он представляет промежуточные данные вычислений. Поток данных изображается в виде стрелки между производителем и потребителем данных, помеченной именами соответствующих данных; примеры стрелок, изображающих потоки данных, представлены на рисунке 2.61. На первом примере изображено копирование данных при передаче одних и тех же значений двум объектам, на втором - расщепление структуры на ее поля при передаче разных полей структуры разным объектам.



*Рис. 2.61. Потоки данных*

*Активные объекты*

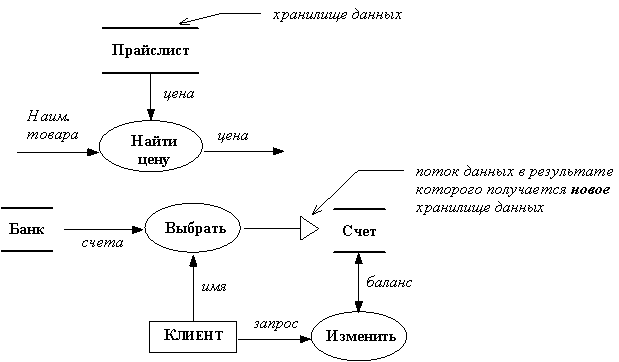
Активным называется объект, который обеспечивает движение данных, поставляя или потребляя их. Активные объекты обычно бывают присоединены к входам и выходам ДПД. Примеры активных объектов показаны на рисунке 2.62. На ДПД активные объекты обозначаются прямоугольниками.



*Рис. 2.62. Активные объекты (экторы)*

*Хранилища данных*

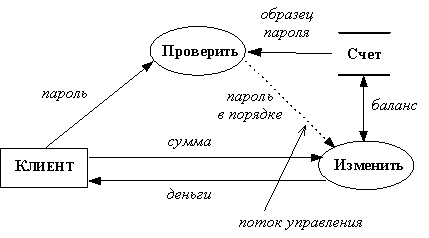
Хранилище данных - это пассивный объект в составе ДПД, в котором данные сохраняются для последующего доступа. Хранилище данных допускает доступ к хранимым в нем данным в порядке, отличном от того, в котором они были туда помещены. Агрегатные хранилища данных, как например, списки и таблицы, обеспечивают доступ к данным в порядке их поступления, либо по ключам. Примеры хранилищ данных приведены на рисунке 2.63.



*Рис. 2.63. Хранилища данных*

*Потоки управления*

ДПД показывает все пути вычисления значений, но не показывает в каком порядке значения вычисляются. Решения о порядке вычислений связаны с управлением программой, которое отражается в динамической модели. Эти решения, вырабатываемые специальными функциями, или предикатами, определяют, будет ли выполнен тот или иной процесс, но при этом не передают процессу никаких данных, так что их включение в функциональную модель необязательно. Тем не менее иногда бывает полезно включать указанные предикаты в функциональную модель, чтобы в ней были отражены условия выполнения соответствующего процесса. Функция, принимающая решение о запуске процесса, будучи включенной в ДПД, порождает в ДПД поток управления (он изображается пунктирной стрелкой).



*Рис. 2.64. Поток управления*

На рисунке 2.64 изображен пример потока управления: клиент, желающий снять часть своих денег со счета в банке, вводит в ATM пароль и требуемую сумму, однако фактическое снятие и выдача денег происходит только в том случае, когда введенный пароль совпадает с его образцом.

Несмотря на то, что потоки управления иногда оказываются весьма полезными, следует иметь в виду, что включение их в ДПД приводит к дублированию информации, входящей в динамическую модель.