



# Модели анализа, проектирования, реализации

РТУ МИРЭА, кафедра ППИ

## СОДЕРЖАНИЕ ЛЕКЦИИ:



*Ахмедова Х.Г. email: [h.ahmedova@mail.ru](mailto:h.ahmedova@mail.ru)*

# Модели основных стаций ЖЦ ИС (согласно УП)

## 1. Формирование (фиксирование) требований

- язык, обычно, естественный/формальный (менее);
- итог – техническое задание ТЗ, календарный план КП;
- основной артефакт - **модель вариантов использования.**

## 2. Анализ (уточнение) требований

- язык - более формальный и специфичный – язык моделирования;
- итог – технический проект (ТП);
- основной артефакт - **модель анализа.**

## 3. Проектирование, реализация (конструирование)

- язык разработчиков (моделирования, программирования)
- итог - версия системы;
- основные артефакты – **модель проектирования, модель реализации)**

## 4. Сопровождение, эксплуатация (переход)

- язык обычно, естественный;
- итог - обученный персонал;
- основной артефакт - **акт приема-сдачи;**

# Процессы ЖЦ ИС (согласно УП )

## Основные технологические процессы:

- приобретение;
- поставка;
- разработка;
- эксплуатация;
- сопровождение.

*интересует  
содержание, а не  
время*

## Вспомогательные технологические процессы:

- документирование;
- управление конфигурацией;
- обеспечение качества;
- разрешение проблем;
- аудит;
- аттестация;
- совместная оценка;
- верификация.

*обеспечивают  
выполнение основных  
процессов*

# Артефакты моделей основных стадий ЖЦ ИС

## На этапе формирования требований:

- создается модель вариантов использования (основное внимание уделяется определению функциональных возможностей (требований) систем)

## На этапе анализа (анализа требований):

- создается модель анализа (основное внимание уделяется уточнению функций, определенных на этапе формирования требований с учетом внутренней организации (архитектуры) проектируемой системы)

## На этапе проектирования:

- создается модель проектирования (основное внимание уделяется детализированному описанию внутренней архитектуры и алгоритмов работы системы)

## На этапе реализации:

- создается модель реализации (основное внимание уделяется логической и физической организации классов в виде **компонентов и подсистем**, а также **топологии** распределенной информационной системы, написанию программного кода)

# Модель формирования требований

Функциональные  
требования - что  
делает система

*регламентируют  
функционирование  
или поведение  
системы (behavioral  
requirements);*

*отвечают на  
вопрос «что  
должна делать  
система» в тех или  
иных ситуациях*

Нефункциональные  
требования - как это  
делает система

*регламентируют  
внутренние и внешние  
условия или атрибуты  
функционирования  
системы;*

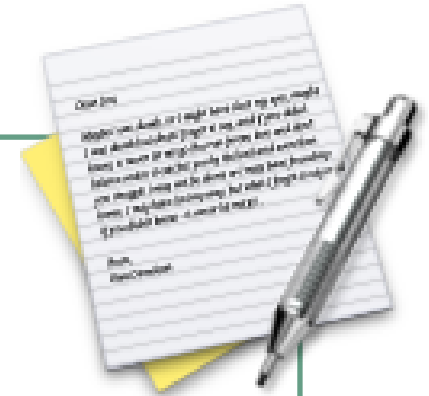
*отвечают на  
вопрос «как это  
делает система»*

# Сбор требований

**Сбор требований** — это один из самых важных этапов процесса создания любой информационной системы.

## Техники сбора требований:

- *анкетирование*
- *интервью*
- *автозапись*
- *изучение существующей документации*
- *повторное использование спецификации (если похожие уже были),*
- *наблюдения за деятельностью и процессами будущих пользователей системы,*
- *мозговой штурм (участники «накидывают» любые идеи, касающиеся решения данной проблемы),*
- *совещание,*
- *эпизод,*
- *и т.д.*



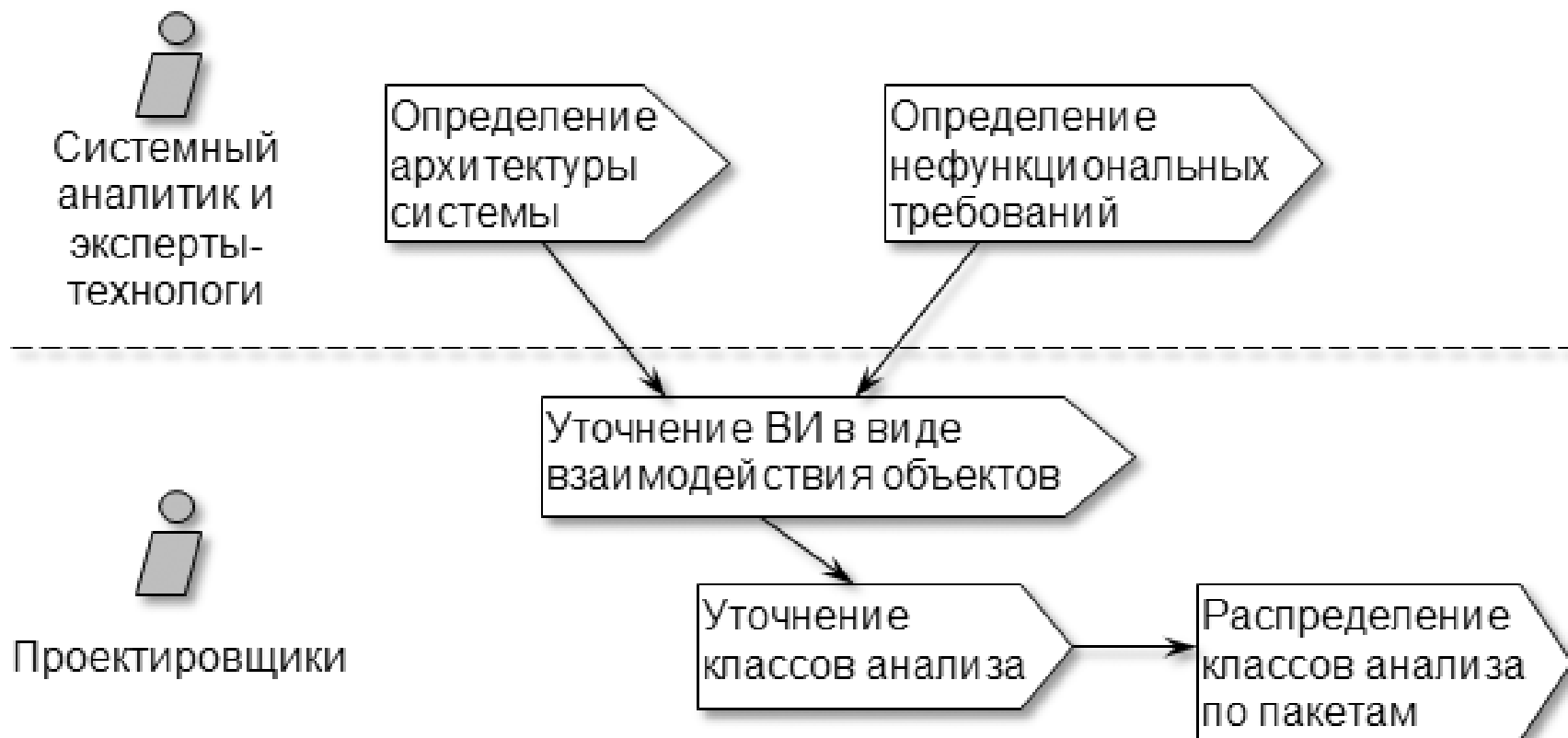
# Модель анализа требований

Для чего строим модель анализа? Что это дает?

- для выявления внутренней архитектуры (определения подсистем и основных классов);
- для поиска альтернативных вариантов реализации системы (подсистем) и выбора основного;
- для уточнения всех требований (функциональных и нефункциональных).



## Алгоритм построения модели анализа (согласно УП)



# Алгоритм составления модели анализа

1. Определяются нефункциональные требования и архитектура системы (состав подсистем и связи между ними, модель предметной области в виде диаграммы классов анализа и т. д.)



2. Для вариантов использования определяется их реализация в виде диаграмм взаимодействия (последовательности и коммуникации).



3. На основе диаграмм взаимодействия уточняется набор классов анализа, связи между ними, атрибуты и методы классов.



4. Выполняется группировка и распределение классов анализа по пакетам.

## Основные артефакты (диаграммы) модели анализа:




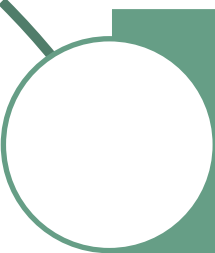
диаграмма классов анализа  
(прообраз классической  
диаграммы классов);

диаграмма  
последовательности;


диаграмма коммуникации  
(кооперации);

диаграмма пакетов.

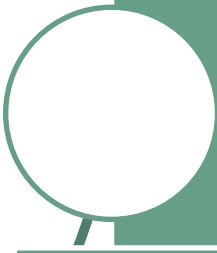
## Дополнительные артефакты модели анализа:




- практичность – характеризуют легкость освоения и эксплуатации системы, интуитивность и эргономичность пользовательского интерфейса, согласованность пользовательского интерфейса, документации и обучающих материалов;



- надежность – характеризуют частоту появления и серьезность ошибок, возможности устранения ошибок и восстановления после сбоев, ремонтпригодность, срок службы и т. д.;



- производительность – накладывают дополнительные ограничения на функциональные требования. Например, возможность параллельного выполнения операций или требования, задающие частоту, скорость, время отклика, выделяемый объем памяти и т. д. для выполнения конкретных операций;



- возможность поддержки – определяют порядок консультаций пользователей в процессе эксплуатации системы, распространения новых версий системы и документации к ней, обновления централизованно распространяемой нормативно-справочной информации и т.д.

## Распределение классов по пакетам позволяет:

- - добиться лучшей структурной организации модели (сильнее формализовать модель);



- - более четко и продуманно распределить обязанности между отдельными разработчиками или их командами;



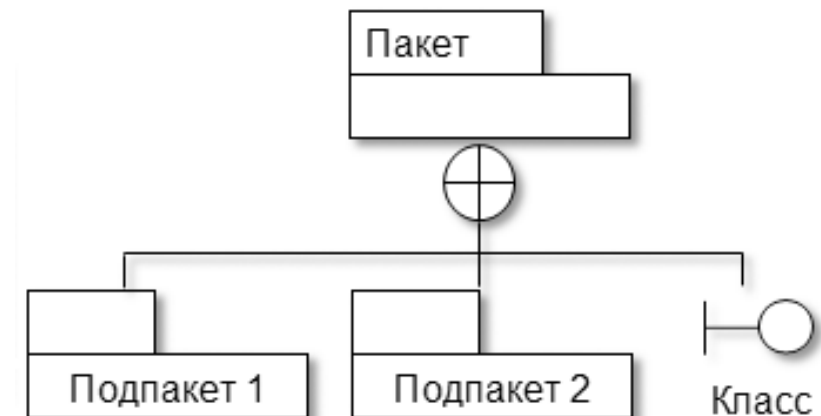
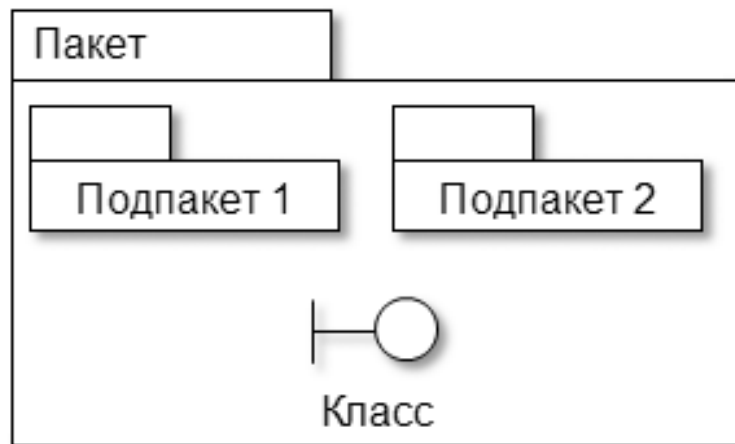
- - упростить повторное использование отдельных пакетов в других проектах, так как связи между пакетами, как правило, минимальны.



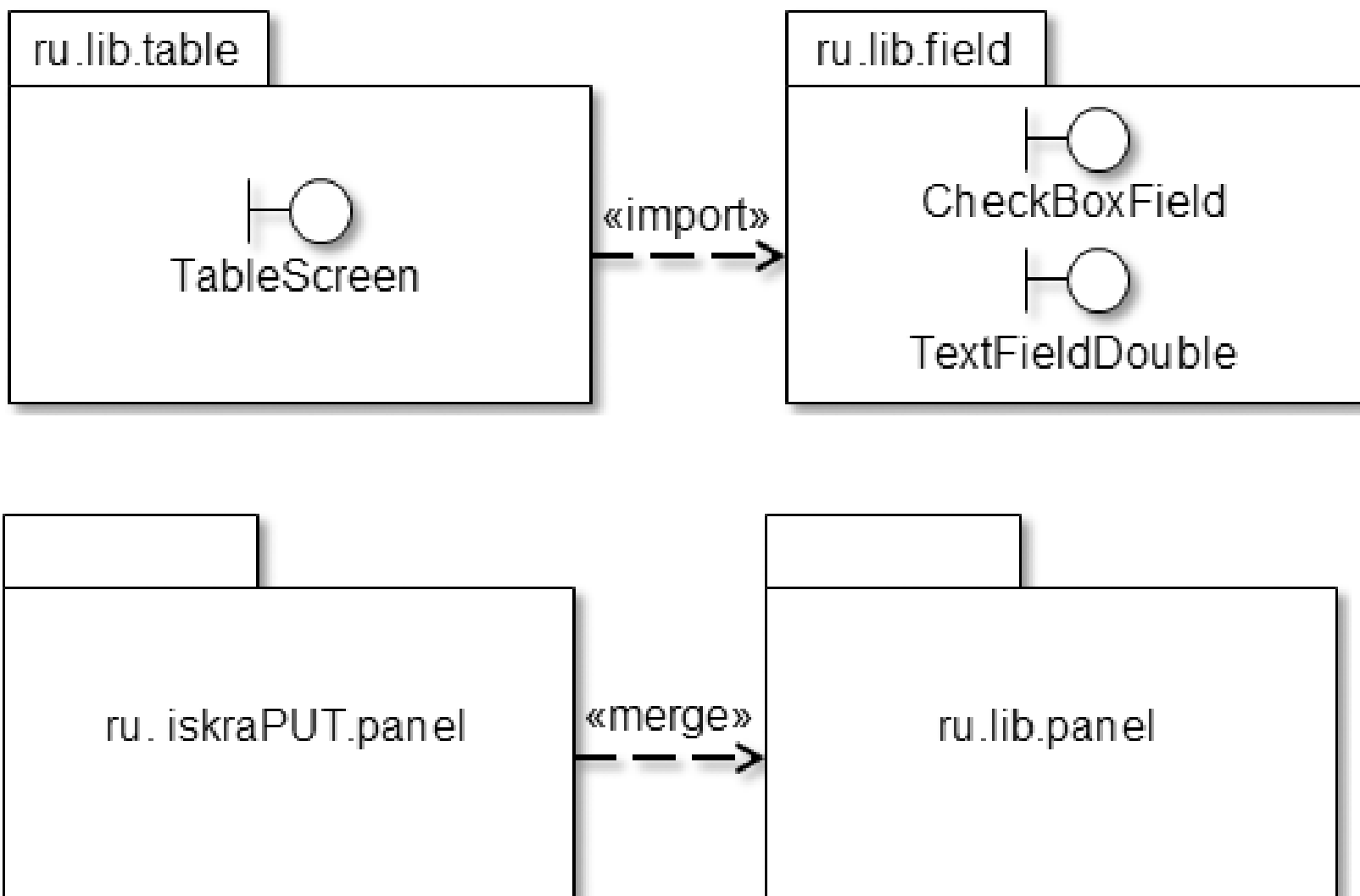
# Способы отображения содержимого пакетов



*стандартное отображение  
пакета*



## Отношение зависимости между пакетами



# Отношение зависимости между пакетами

## import

- позволяет при обращении сущности из одного пакета к сущности другого пакета указывать только ее имя, а не полную спецификацию;
- *например, если в объекте класса `TableScreen` необходимо создавать объекты класса `TextFieldDouble`, после импортирования достаточно для этих целей использовать имя класса «`new TextFieldDouble(...)`» вместо «`new ru.lib.field.TextFieldDouble(...)`».*

## merge

- практически идентична отношению обобщения. В частности, пакет `ru.iskraPUT.panel` помимо своих сущностей (подпакетов и классов) будет содержать сущности пакета `ru.lib.panel`;
- *если в двух пакетах будут сущности с одинаковым именем, то сущность в результирующем пакете (`ru.iskraPUT.panel`) будет расширена за счет специфических свойств сущности исходного пакета (`ru.lib.panel`);*
- *отличие от отношения обобщения заключается в отсутствии наследования сущностей с областью видимости «`private`» (частных сущностей).*



# Группировка классов по пакетам: подходы

группировка классов по стереотипу



- *в одном пакете будут находиться классы сущностей, в другом – граничные, в третьем – управляющие;*

группировка классов по семантической однородности



- *например, пакет «Безопасность» будет содержать все классы, отвечающие за безопасность системы;*

группировка классов по подсистемам (по функциональности)



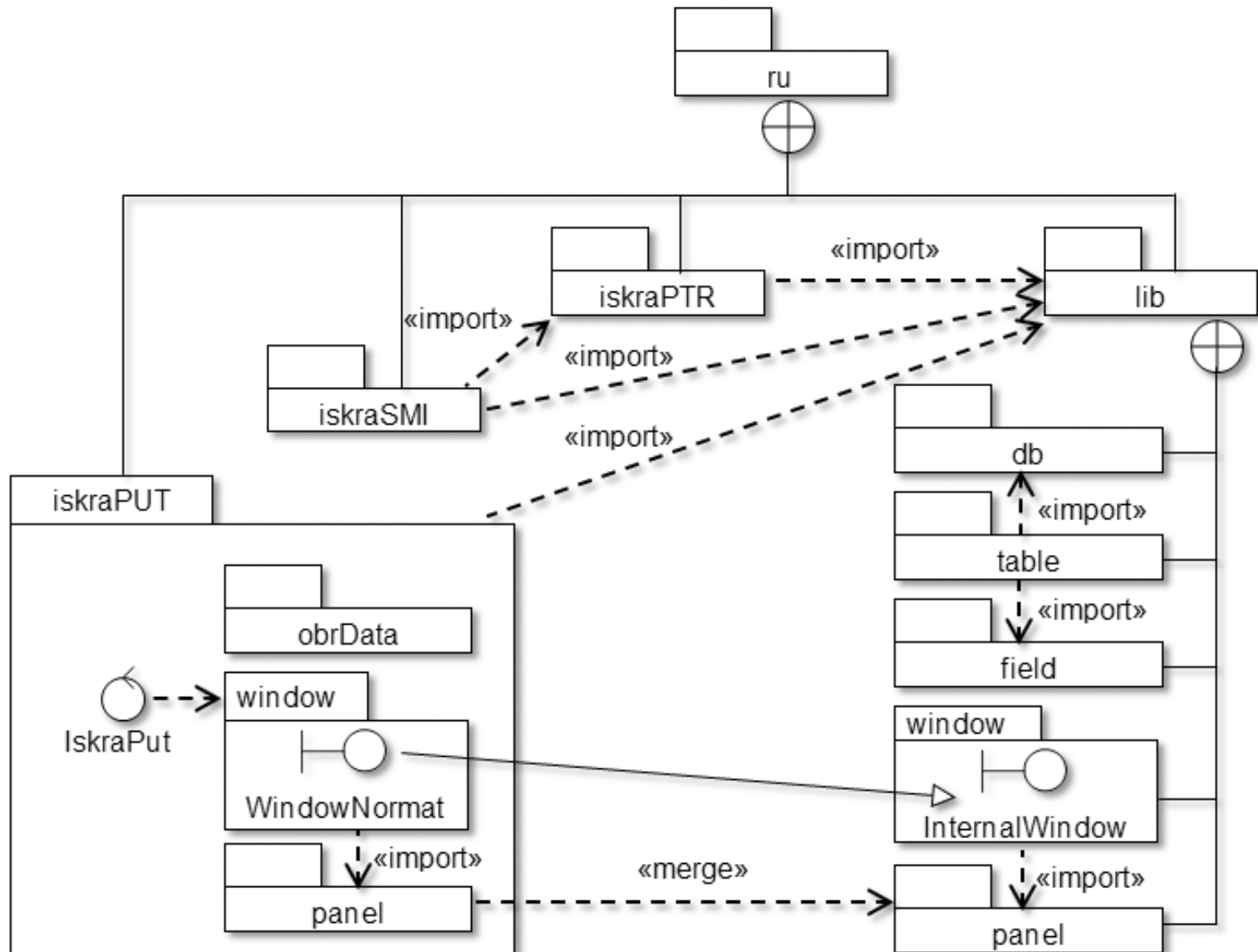
- *например, при разработке большой и сложной системы, состоящей из нескольких взаимодействующих подсистем;*

комбинация разных подходов



- *например, одновременно по стереотипу, по функциональности и семантической однородности на разных уровнях*

## Пример комбинации подходов



# Модели проектирования

- **Модель проектирования** предназначена для создания полного детализированного описания внутренней архитектуры и алгоритмов работы системы.



- Модель проектирования рекомендуется разрабатывать без привязки к конкретным языкам программирования, с помощью которых будет создаваться программный продукт, т. е. разрабатывать *логическую модель*.

*\*\*\*На самом деле создать модель без оглядки на используемые языки программирования невозможно, но, по крайней мере, необходимо стремиться к этому.*

# Модель проектирования:

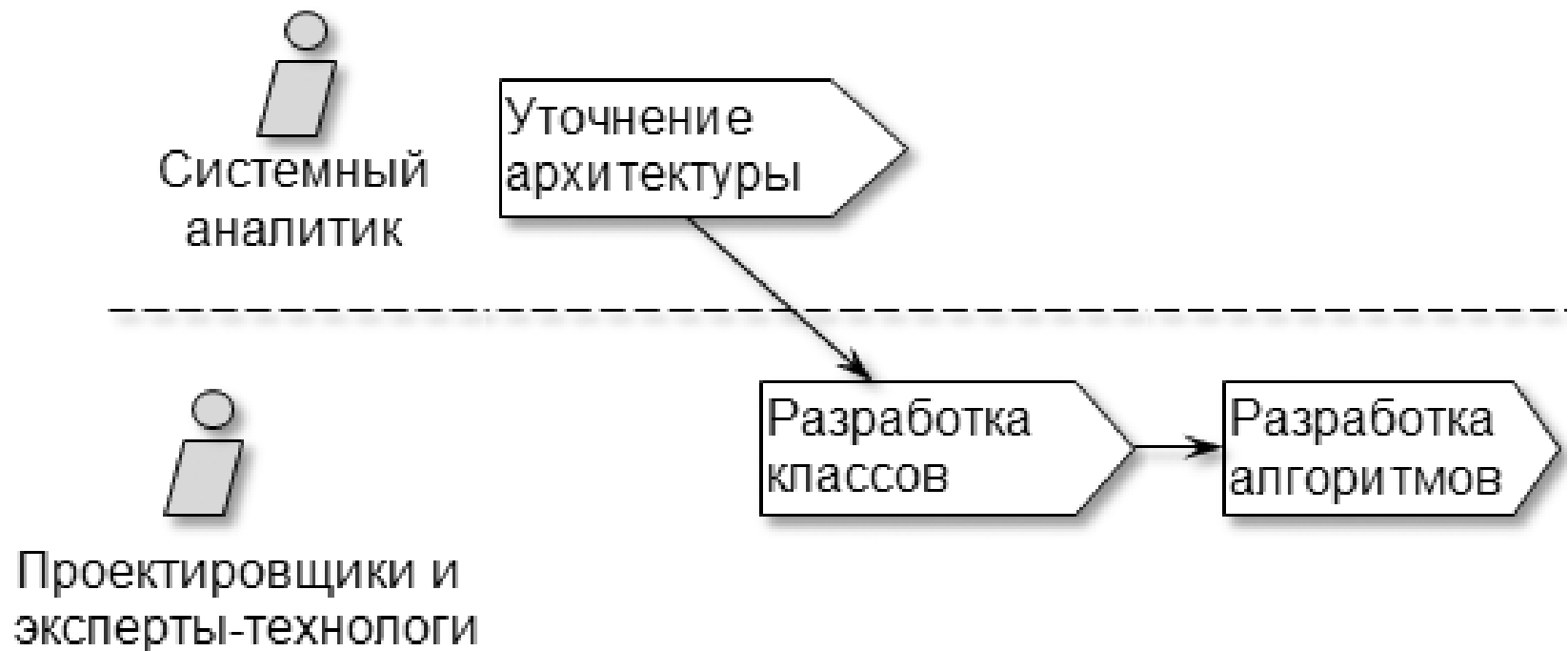
*Для чего строим модель проектирования?  
Что это дает?*

- для уточнения внутренней архитектуры и вариантов использования системы;

- для уточнения требований;

- для определения детализированных алгоритмов работы системы в целом и ее отдельных элементов.

## Алгоритм построения модели проектирования (согласно УП)



# Основные артефакты модели проектирования:

*В  
разработке  
алгоритмов,  
специфичных для  
предметной  
области,  
непосредственное  
участие должны  
принимать  
эксперты-  
технологи.*

**Диаграмма классов** (основное внимание уделяется проектированию классов (их атрибутов и операций), компоновке классов в подсистемы и определению интерфейсов между классами и подсистемами).

**Диаграмма деятельности** (основное внимание уделяется детальному описанию операций и взаимодействию между классами с помощью деятельности, описывающих алгоритмы работы).

# Диаграмма состояния

**Диаграмма автоматов** представляет собой связный ориентированный граф, вершинами которого являются состояния, а дуги служат для обозначения переходов из состояния в состояние.

**Под состоянием** (англ. state) понимается ситуация в ходе жизни экземпляра сущности, когда эта ситуация удовлетворяет некоторому условию, экземпляр выполняет некоторые операции или ждет наступления некоторого события.

*Например, для объекта его состояние может быть задано в виде набора конкретных значений атрибутов, при этом изменение этих значений будет приводить к изменению состояния моделируемого объекта.*

# Виды операций: действие и деятельность

## Действие (англ. action)

- – это атомарная операция, выполнение которой не может быть прервано, приводящая к смене состояния или возвращающая значение.
- *Примерами действий служат операции создания или уничтожения объекта, расчет факториала и т. д.*

## Деятельность (англ. activity)

- – это составная (неатомарная) операция, реализуемая экземпляром в конкретном состоянии, выполнение которой может быть прервано.
- *В частности, под деятельностью можно понимать процедуры расчета допускаемых скоростей или шифрования данных.*



# События

– это спецификация существенного факта, который может произойти в конкретный момент времени.

Событие (англ. event )



## Внутренние события

- *передаются между объектами внутри системы*

## Внешние события

- *передаются между системой и актерами (например, нажатие кнопки или посылка сигнала от датчика передвижений)*

# Виды событий в UML

посылка сообщения (англ. message):

вызов (англ. call);

сигнал (англ. signal);

любое сообщение (англ. any receive);

событие времени (англ. time);

изменение состояния (англ. change).

# Моделирование событий в UML

**Вызов** – спецификация факта послылки синхронного сообщения между объектами, предписывающего выполнение операции (действия или деятельности) объектом, которому посылается сообщение.

**Сигнал** – спецификация факта послылки асинхронного сообщения между объектами. Исключения, которые поддерживаются в большинстве современных языков программирования, являются наиболее распространенным видом внутренних сигналов.

**Событие времени** – спецификация факта, обозначающего наступление конкретного момента времени (англ. absolute time) или истечение определенного промежутка времени (англ. relative time). В UML данный факт обозначается с помощью ключевых слов «at» (например, at 9:00:00) и «after» (например, after 2 seconds).

**Изменение состояния** – спецификация логического условия, соответствующего изменению состояния экземпляра сущности. В UML оно обозначается с помощью ключевого слова «when» (например, when  $A < B$ ) или сторожевого условия (например,  $[A < B]$ ).

## Элемент диаграммы «Состояние»

- **Состояние** отображается в виде четырехугольника со скругленными углами, внутри которого обязательно записывается имя.
- Рекомендуется в качестве имени использовать глаголы в настоящем времени (звонит, печатает, ожидает) или причастия (занято, передано, получено).

### Способы отражения состояний



# Характеристика состояния: метки

## - entry (англ. – вход)

- – действие при входе, выполняемое вне зависимости от того, по какому переходу был выполнен вход в состояние;
- *например, создать соединение с базой данных entry / createConnect();*

## - exit (англ. – выход)

- – действие при выходе, выполняемое вне зависимости от того, по какому переходу был выполнен выход из состояния;
- Например, закрыть соединение с базой данных exit / closeConnect();

## - do (англ. – выполнять)

- – деятельность в состоянии (бездействие и ожидание наступления некоторого события, или выполнение длительной операции);
- например, рассчитать допускаемые скорости do / calculateVdop();
- допускается указывать несколько операций в виде отдельных строк, каждая из которых начинается с метки «do», или в виде одной строки, операции в которой отделены друг от друга точкой с запятой.

## Пример состояния с характеристикой

### Определение допускаемых скоростей

```
entry / createConnect()  
do / loadData()  
do / calculateVdop()  
do / saveData()  
newTarget / pauseCalculateVdop()  
defer / showDataError()  
exit / closeConnect()
```

## Элемент диаграммы «Переход»

- ❖ **Переход** (англ. transition) – отношение между двумя состояниями, показывающее возможный путь изменения состояния экземпляра сущности.
- ❖ Считается, что в состоянии экземпляр сущности находится продолжительное время, а переход выполняется мгновенно.
- ❖ Переход отображается в виде однонаправленной ассоциации между двумя состояниями.
- ❖ При смене состояний говорят, что переход срабатывает: до срабатывания перехода экземпляр сущности находится в исходном состоянии, после его срабатывания – *в целевом состоянии*.

# Виды переходов в диаграммах состояния (автоматов)

- срабатывает неявно, когда все основные операции (с метками entry, do и exit) в исходном состоянии успешно завершают свою работу;
- обозначается стрелкой без надписи.

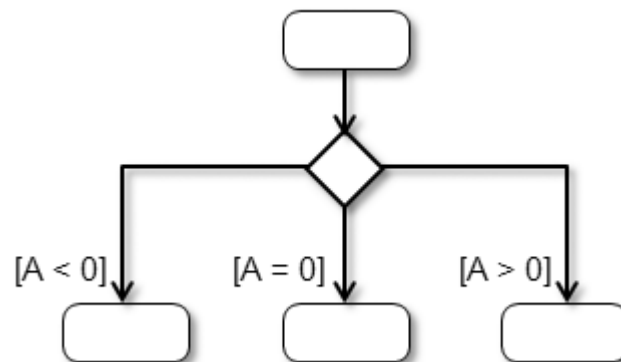
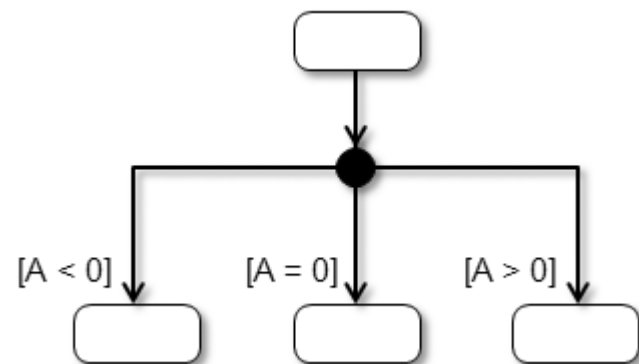
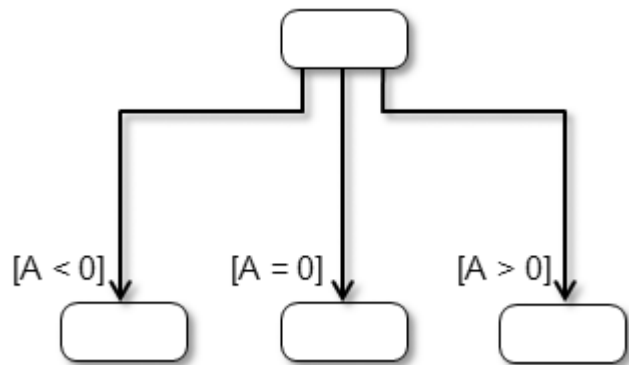
**Нетриггерный  
переход**  
(переход по  
завершению)

- необходимо наступление некоторого события, которое записывается над стрелкой;
- над стрелкой может быть записана строка текста вида «*событие [сторожевое условие] / действие*»

**Триггерный  
переход**



# Способы отображения триггерных переходов



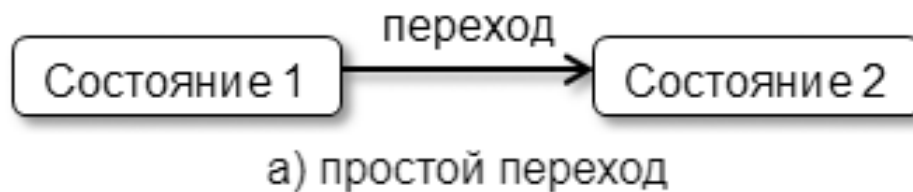
## Примеры спецификации переходов:

- **mouseClick()** – нажатие кнопки мыши в момент, когда указатель находится над моделируемым объектом (например, над командной кнопкой, запускающей процедуру определения допускаемых скоростей);
- **mouseClick()** / **setFocus()** – нажатие кнопки мыши с одновременным установлением фокуса на моделируемом объекте;
- **mouseClick() [isEnabled()] / setFocus()** – нажатие кнопки мыши с одновременным установлением фокуса на моделируемом объекте при условии, что он доступен.
- при построении концептуальных диаграмм допускается обозначать переход произвольной строкой текста, характеризующей событие. Например, «столкновение» или «выход из строя».

## Примеры переходов

**Рефлексивный переход** - переход направлен в то же состояние, из которого он выходит.

В отличие от внутренних переходов, при рефлексивном переходе выполняются внутренние действия, ассоциированные с метками entry и exit.



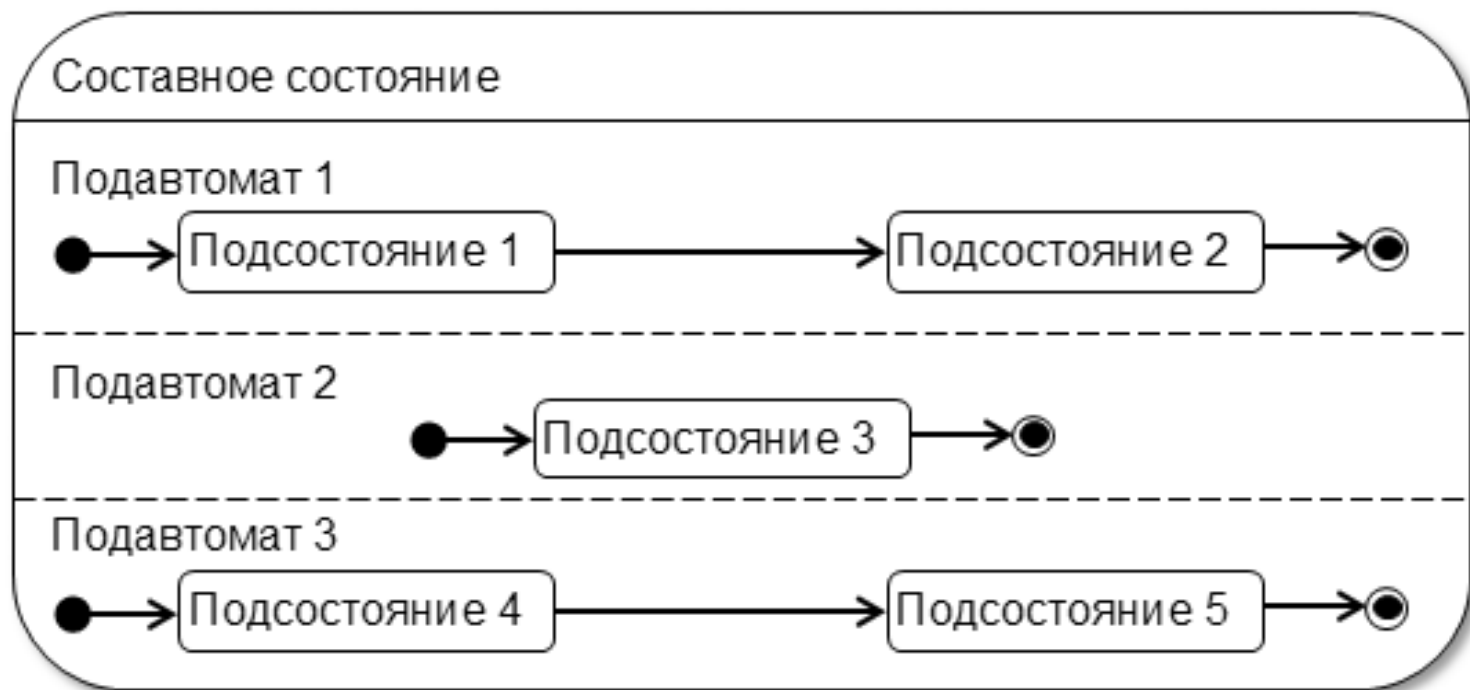
б) рефлексивный переход

## Составное состояние и подсостояния



## Составное состояние с вложенными параллельными подавтоматами

- Составное состояние, которое может использоваться в разных контекстах, в т.ч. и для разных диаграмм (автоматов), называются **подавтоматами** (англ. submachine state).
- Составное состояние может быть разбито на **зоны** (англ. regions), называемые также **параллельными подавтоматами** (англ. concurrent substates).



## Пример составного состояния со скрытой внутренней структурой



# Псевдосостояния

Наименование	Обозначение	Назначение
Начальное (англ. initial)	●	Начальное состояние автомата, начальное подсостояние составного состояния или параллельного подавтомата. Из начального состояния могут исходить переходы.

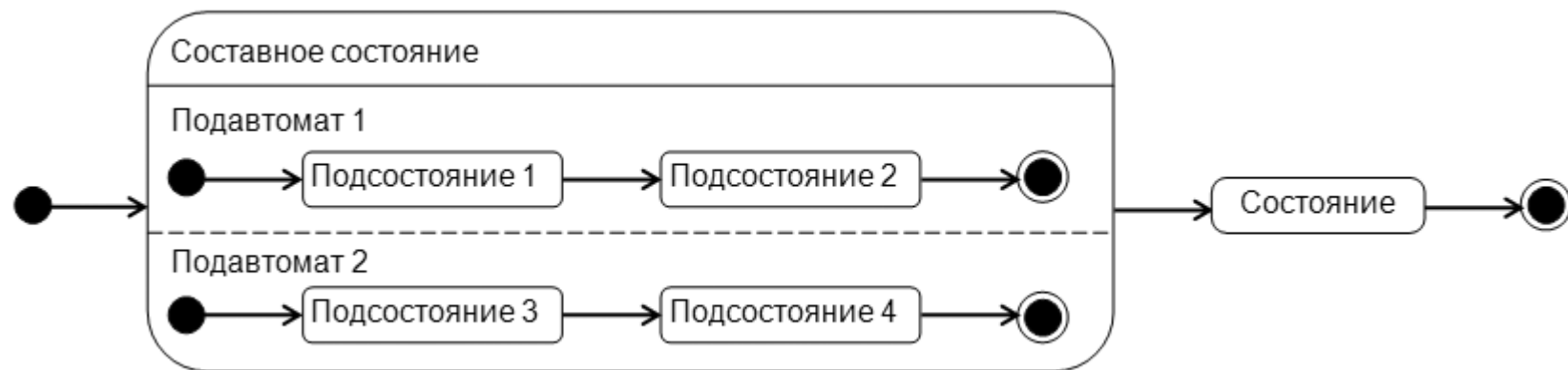
Конечное  
(англ. final)



Конечное состояние автомата, конечное подсостояние составного состояния или параллельного подавтомата. В конечном состоянии могут только входить переходы. Примечание. В стандарте UML 2.5 считается состоянием, а не псевдосостоянием.

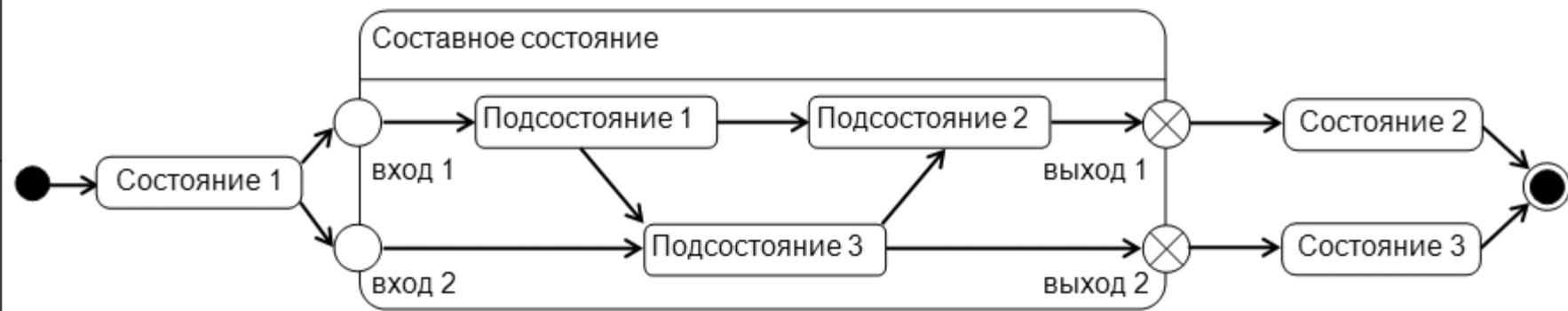


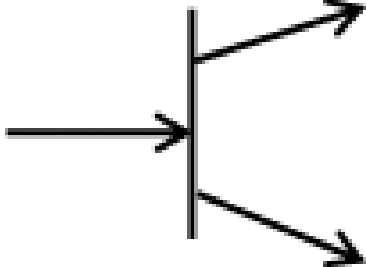
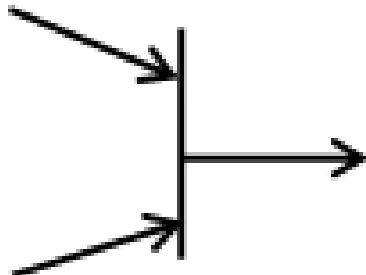
# Пример

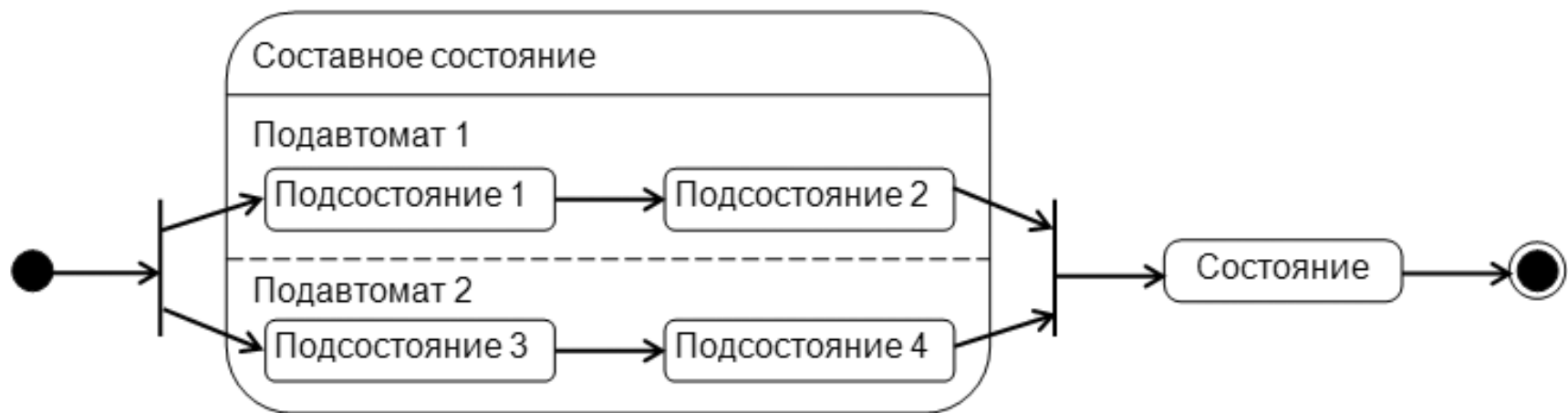



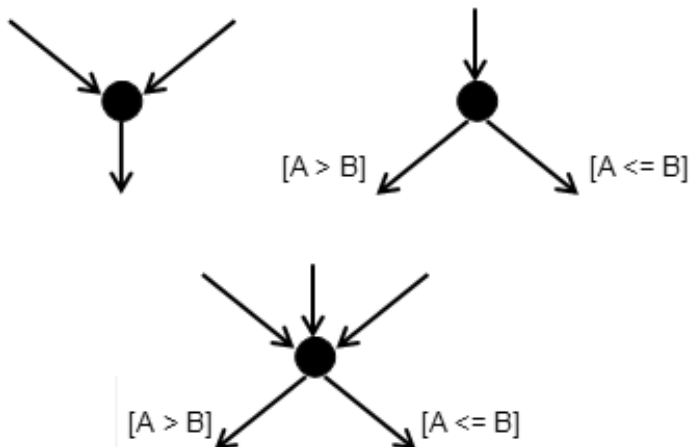

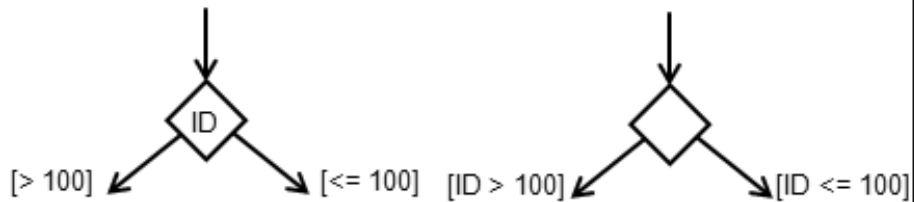
Немедленное завершение (англ. terminate)	X	Аналогично конечному состоянию, но подразумевает немедленное прекращение деятельности и уничтожение экземпляра сущности, для которой построен автомат.
---	---	--

Точка входа (англ. entry point)		Точка входа в автомат или составное состояние. Может быть несколько. Допускается крепление к границе составного состояния.
Точка выхода (англ. exit point)		Точка выхода из автомата или составного состояния. Может быть несколько. Допускается крепление к границе составного состояния.



Ветвление (англ. fork)		Ветвление переходов параллельные подавтоматы.	в
Соединение (англ. join)		Соединение переходов параллельных подавтоматов. Выполняет функцию синхронизации выхода параллельных подавтоматов составного состояния.	из       из



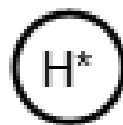
<p>Перекресток (англ. junction)</p>		<p>Соединение и ветвление переходов для последовательных состояний. В случае ветвления для каждой исходящей из перехода ассоциации должно быть задано сторожевое условие.</p>	
<p>Выбор (англ. choice)</p>		<p>Аналогично переходу, работающему на ветвление для последовательных состояний.</p>	

- Поверхностное историческое (англ. shallow history)



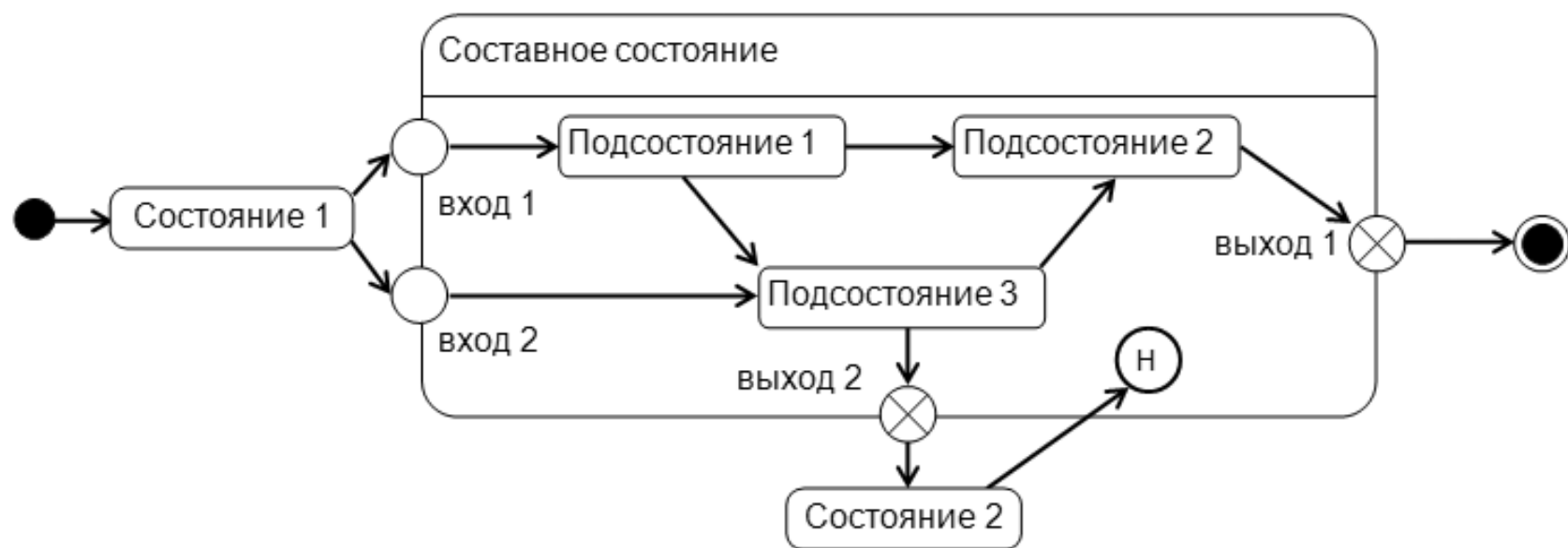
Указывается внутри составного состояния и подразумевает запоминание текущей конфигурации составного состояния при выходе из него. Переход в историческое состояние восстанавливает запомненную конфигурацию составного состояния и продолжает работу составного состояния с того момента, когда его прервали в прошлый раз. Внутри составного состояния может быть только одно историческое состояние.

- Глубинное историческое (англ. deep history)



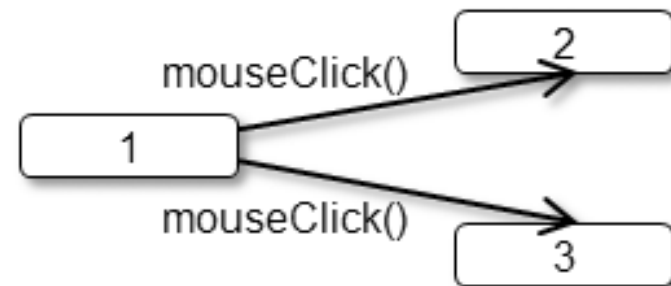
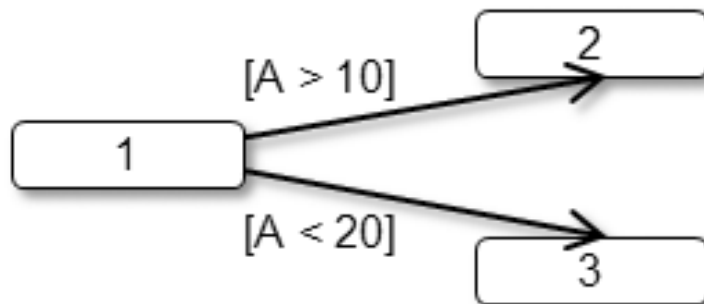
Аналогично поверхностному историческому состоянию, но распространяется на все уровни вложенности подсостояний.





## Ошибки в диаграмме

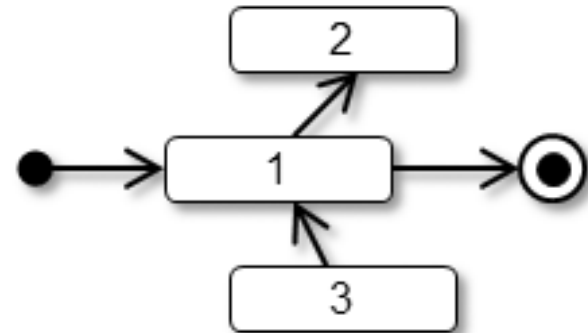
- В каждый момент времени автомат или подавтомат должен находиться только в одном состоянии. Это означает, что спецификация переходов из одного состояния не должна допускать потенциальной возможности перехода в два и более состояний.
- На следующем рисунке приведены примеры фрагментов диаграмм с конфликтными переходами.



Исключением из этого правила является параллельный переход в подсостояния параллельных подавтоматов одного составного состояния.

# Правила разработки диаграммы состояния

- 1. Наличие у экземпляра сущности нескольких состояний, отличающихся от простой схемы «исправен – неисправен» или «активен – неактивен», служит признаком необходимости построения диаграммы автоматов.
- 2. Автомат (диаграмма) обязательно должен начинаться знаком начального состояния и заканчиваться знаком конечного.
- Начальное состояние указывается только один раз, а конечных может быть несколько в целях минимизации пересечений переходов.
- 3. Диаграмма не должна содержать изолированных состояний и переходов. Переходы и их спецификация должны быть заданы таким образом, чтобы на графе каждое состояние было потенциально достижимо из начального и из любого состояния было потенциально достижимо конечное.



*На рисунке из состояния 2 нет пути в конечное, а из начального состояния нет пути в состояние 3.*

# Диаграмма деятельности

- **Диаграмма деятельности** – диаграмма детализирующая особенности алгоритмической и логической реализации выполняемых системой операций.
  - акцентирует внимание на последовательности выполнения определенных действий, которые в совокупности приводят к получению желаемого результата.
  - графически, представляется в виде ориентированного графа, вершинами которого являются действия или деятельности, а дугами – переходы между ними.
  - могут быть построены для отдельного варианта использования, кооперации, метода и т. д.
  - являются разновидностью диаграмм автоматов, но если на диаграмме состояний основное внимание уделяется статическим состояниям, то на диаграмме деятельности основное внимание уделяется действиям.

## Основные элементы диаграммы деятельности:

- исполняемые узлы;

- объекты;

- переходы;

- управляющие узлы;

- коннекторы;

- группирующие элементы.

## Исполняемые узлы

- **К исполняемым узлам** относятся *действия* (англ. action) и *деятельности* (англ. activity).
- На **блок-схемах** их аналогами являются процессы и predetermined процессы.
- Обычное использование исполняемых узлов заключается в моделировании одного шага выполнения алгоритма (процедуры) или потока управления.
- Графически исполняемые узлы отображаются, как простые и составные состояния.

$S := \text{Height} * \text{Width}$

а) действие

Разработать план проекта



б) деятельность

- Внутри фигуры записывается выражение действия (англ. action expression), записываемое на естественном языке, некотором псевдокоде или языке программирования.

# Объекты

**К объектам** относятся:

- непосредственно объекты (англ. object) в традиционном понимании UML;
- отправка сигнала (англ. send signal);
- прием сигнала (англ. accept signal);
- событие времени (англ. time event).



а) объект



б) посылка сигнала



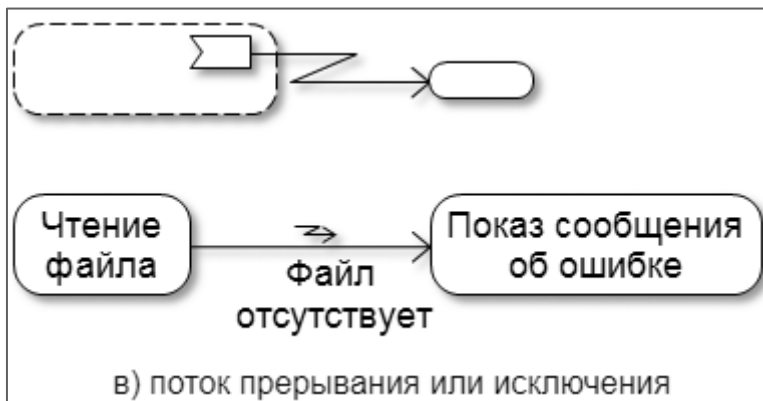
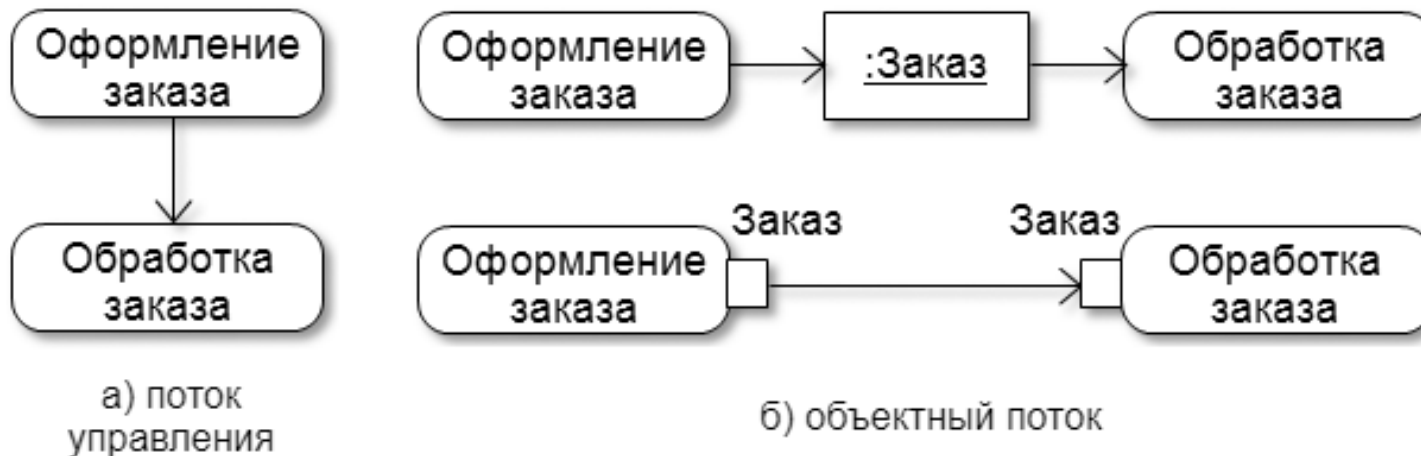
в) прием сигнала



г) событие времени

# Переход

**Переход** (англ. transition или activity edge), отображается ассоциацией.

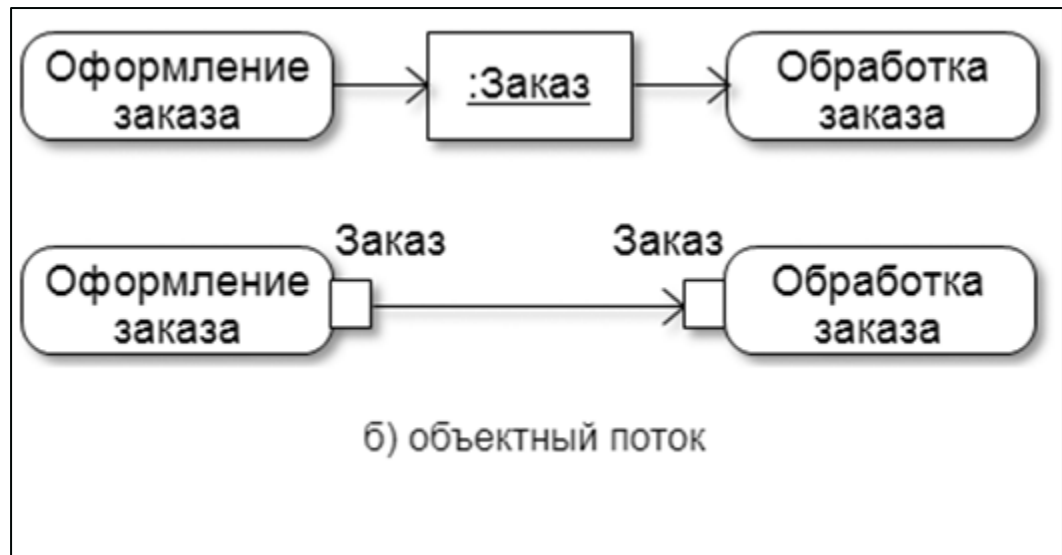
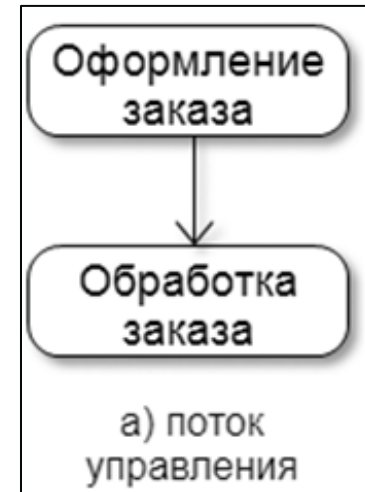




# Виды переходов

- **Поток управления** (англ. control flow) представляет собой самый общий вид перехода и задает порядок выполнения операций.

**Объектный поток** (англ. object flow) переход, который помимо передачи управления позволяет отобразить и передачу информации. (ассоциации соединяются с символом «объекта» или специальными контактами (англ. pins), прикрепленными к границам действий).



## Виды переходов

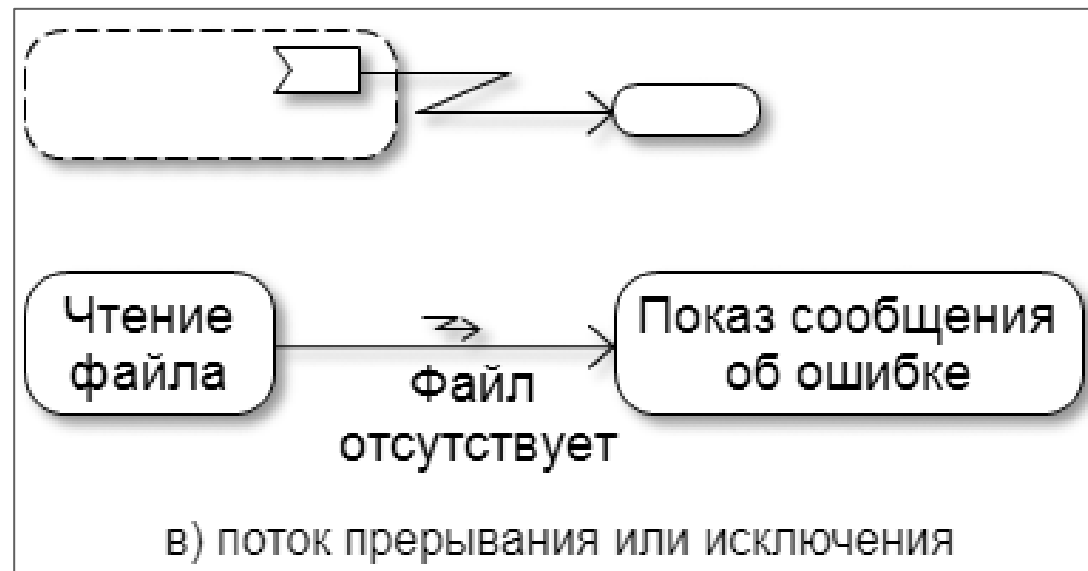
- **Поток прерывания** (англ. interruptible flow), как правило, исходит из символа «прием сигнала», расположенного в прерываемой области, и входит в действие - обработчик прерывания.

### **Поток**

#### **исключения**




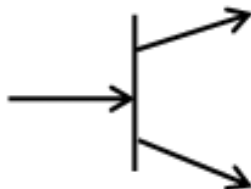
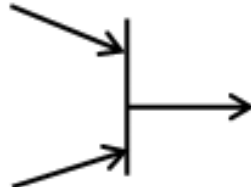

(англ.  
exception  
flow)

используется так же, как и поток прерывания. Отличие прерывания от исключения состоит в том, что первое - это допустимое альтернативное событие в системе, а второе - ошибка при выполнении действия.



# Управляющие узлы

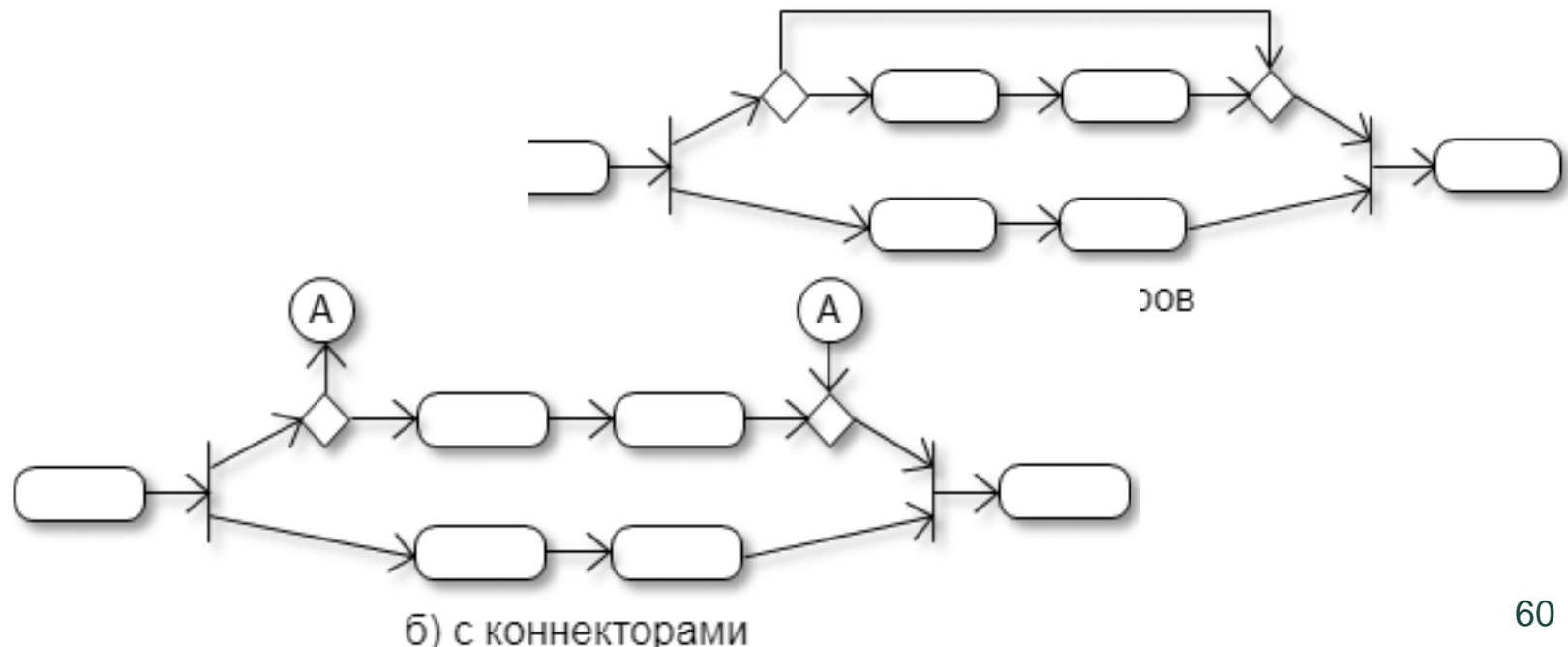
- **Управляющим узлам** (англ. control nodes) на диаграмме деятельности соответствуют псевдосостояния на диаграмме автоматов

Обозначение	Наименование	
	на диаграмме автоматов	на диаграмме деятельности
	Начальное псевдосостояние (англ. initial pseudostate)	Начальный узел (англ. initial node)
	Конечное состояние (англ. final state)	Завершение деятельности (англ. activity final)
	Точка выхода (англ. exit point pseudostate)	Завершение потока (англ. flow final)
	Ветвление (англ. fork pseudostate)	Ветвление (англ. fork node)
	Соединение (англ. join pseudostate)	Соединение (англ. join node)
	Выбор (англ. choice pseudostate)	Слияние / решение (англ. merge / decision node)

# Коннекторы

**Коннекторы** (англ. connectors) выступают в качестве соединителей, применяемых на блок-схемах.

**Коннекторы** используются для прерывания потока в одной части диаграммы и продолжении в другой, если диаграмма занимает несколько листов или отображение потока перенасыщает диаграмму. Коннектор представляется в виде круга, внутри которого пишется его идентификатор.



## Группирующие элементы

**Разделы деятельности** обычно используют для моделирования бизнес-процессов или совместной работы нескольких сущностей (актеров, объектов, компонентов, узлов и т.д.)

- в этом случае диаграмма делится на разделы (области) вертикальными или горизонтальными линиями, в заголовке которых указываются имена сущностей, ответственных за выполнение действий внутри соответствующего раздела.

**Прерываемый регион** группирует действия, обычная последовательность выполнения которых может быть прервана в результате наступления нестандартной ситуации (например, при оформлении кредита клиент от него отказывается).

- отображается четырехугольником со скругленными углами и штриховым контуром.

## Рекомендации по разработке диаграмм деятельности

1. При построении диаграмм рекомендуется использовать классические принципы моделирования – декомпозиции и иерархического упорядочения. Т.е. при моделировании алгоритма вначале строится контекстная диаграмма с деятельностями, которые после детализируются с помощью соответствующих диаграмм декомпозиции.
2. Количество пересечений линий следует минимизировать. При этом считается, что пересекающиеся линии не имеют логической связи друг с другом. Другими словами потоки данных или управления в местах пересечений не меняют своего направления.
3. Если на диаграмме имеется ветвление/решение на параллельные или альтернативные потоки, то должно указываться и соответствующее соединение/слияние этих потоков.
4. При использовании альтернативных потоков каждый из них должен быть специфицирован с помощью сторожевого условия. Сторожевые условия не должны допускать одновременного срабатывания двух и более переходов.
5. В целях определения зоны ответственности (набора действий) сущностей рекомендуется использовать разделы деятельности.



# Спасибо за внимание

РТУ МИРЭА, кафедра ППИ