Системы реального времени

ОСРВ - Архитектура и проектирование

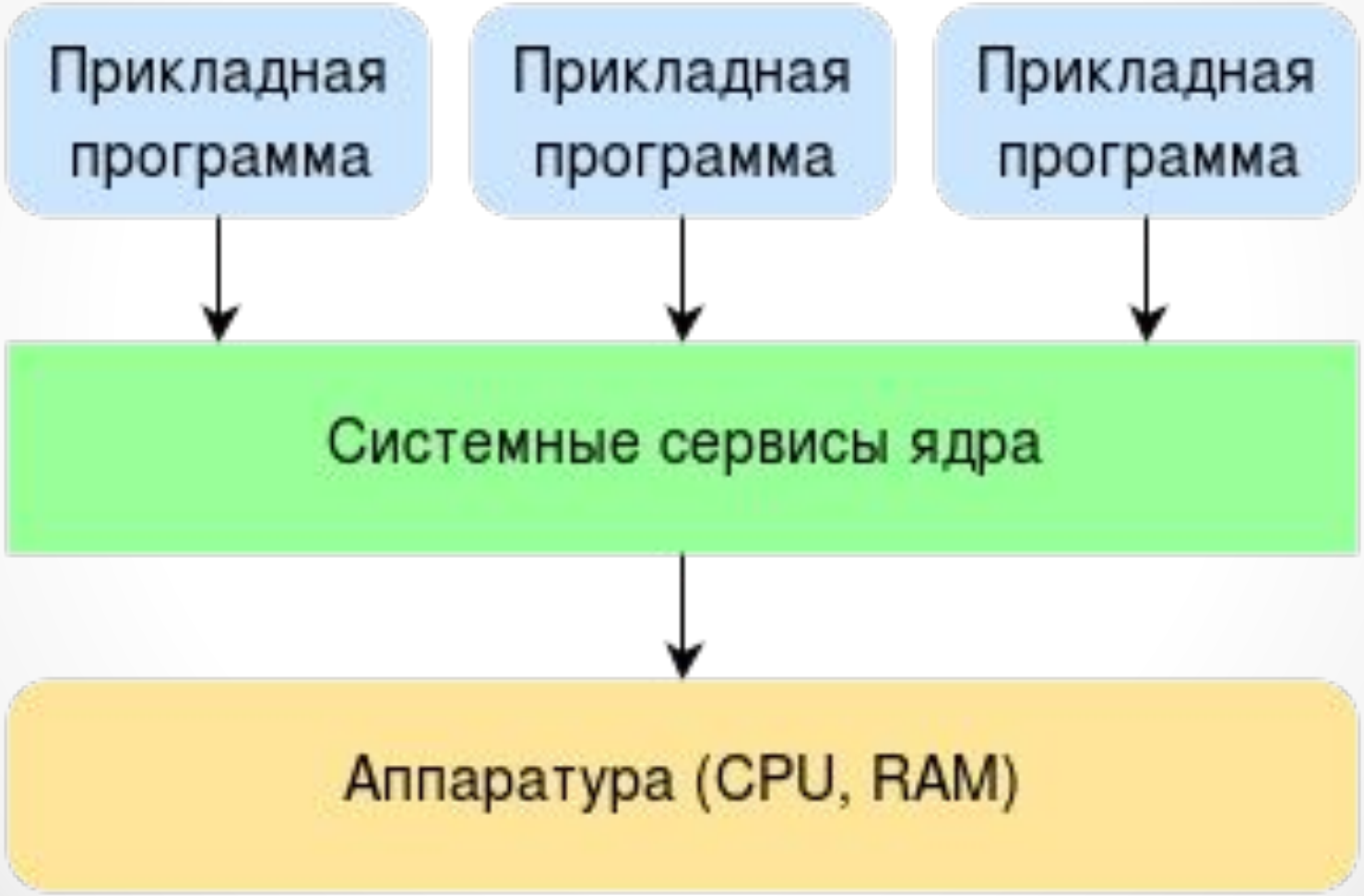
OCPB

- **VxWorks**
- **QNX Neutrino RTOS**
- **RTEMS**
- **ChorusOS**
- **TinyOS**
- **OSEK/VDX**
- **OSE RTOS**
- **Free RTOS**
- **Contiki**
- **pSOS**
- **INTEGRITY**
- **LynxOS**
- **Microware OS-9**
- **GRACE-OS**
- **C EXECUTIVE**
- **CMX-RTX**

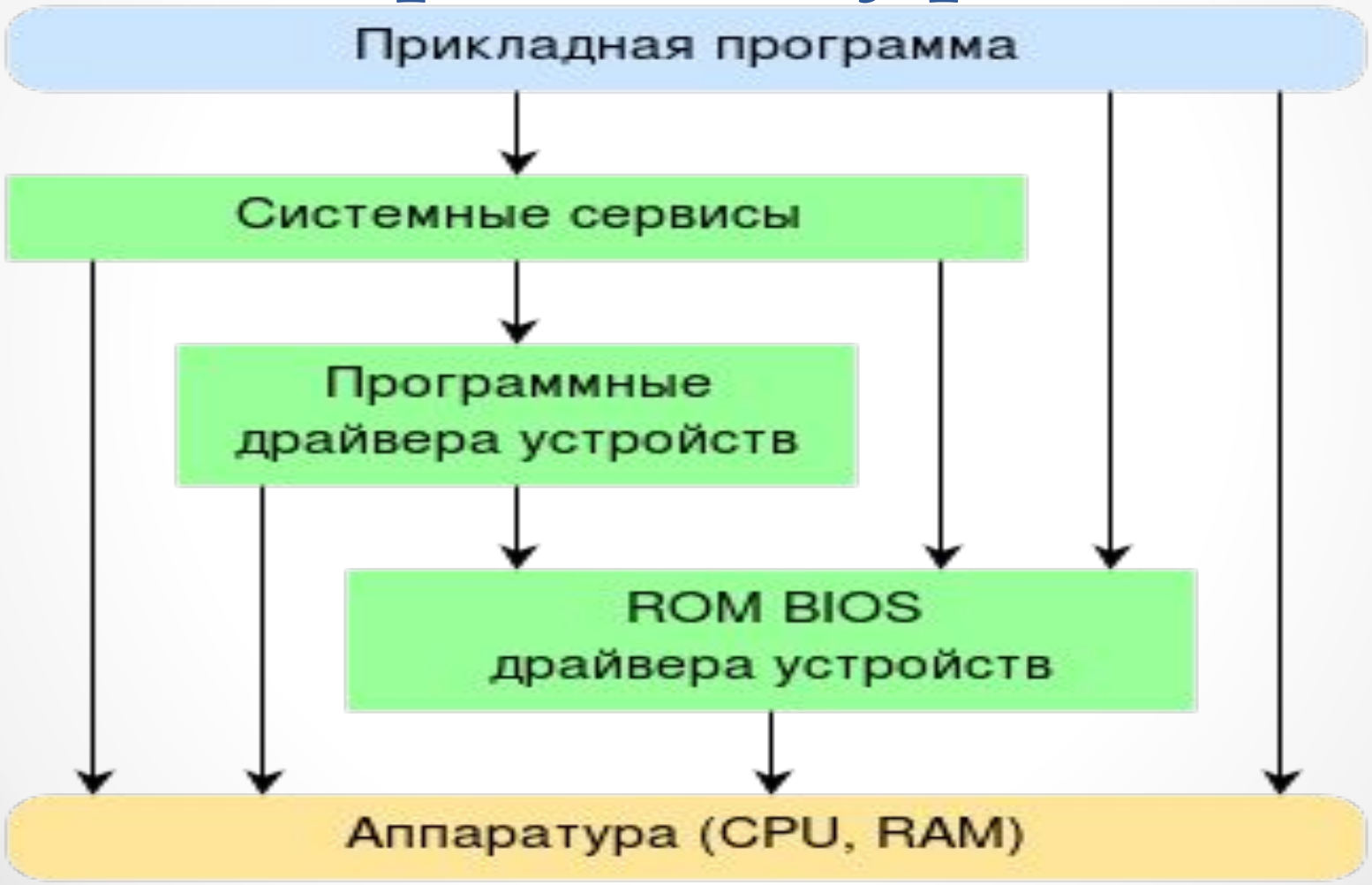
Структура ОС РВ

- Монолитная ОС РВ
- Многослойная ОС РВ
- Клиент-скерверная ОС РВ

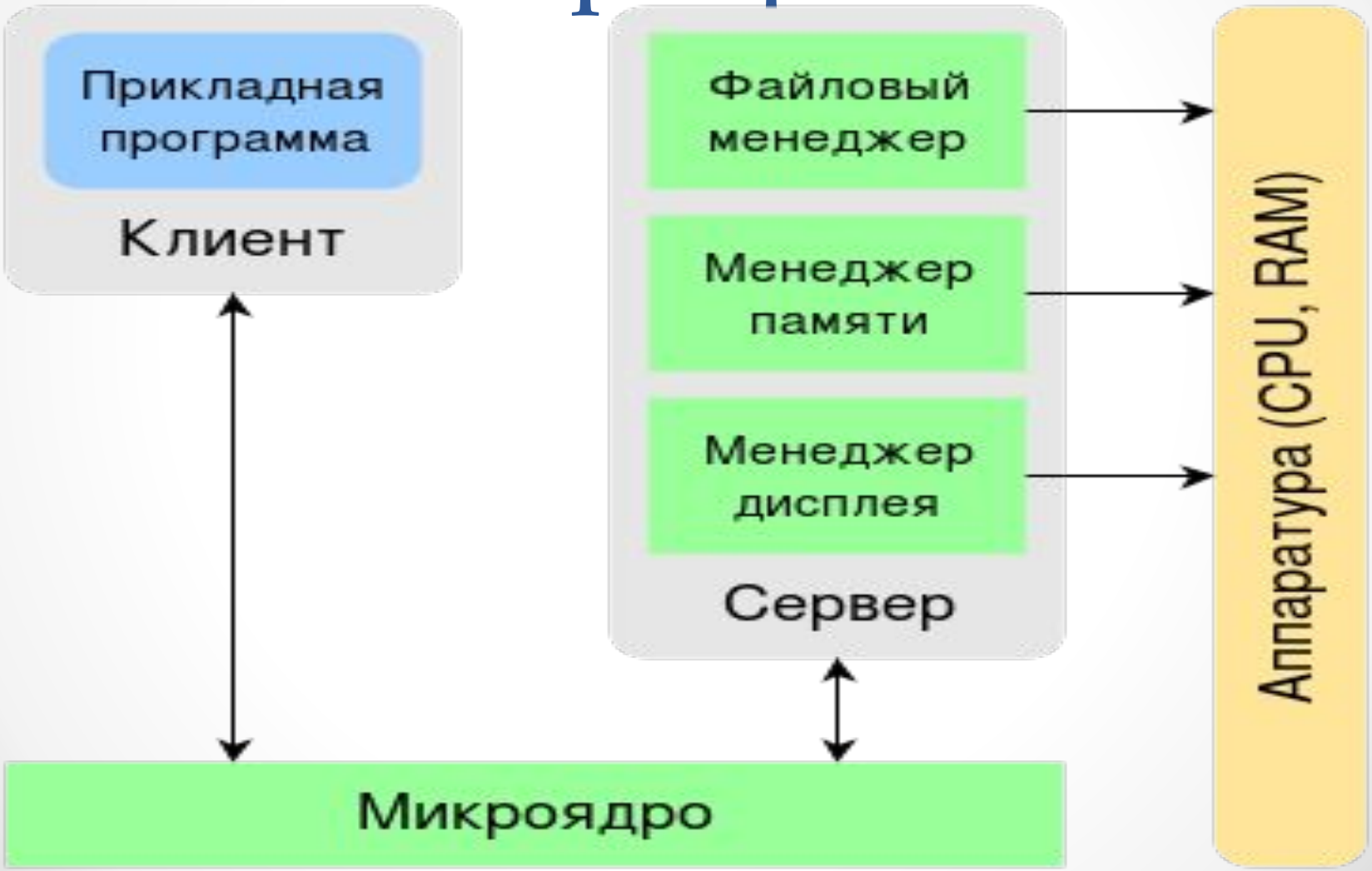
Монолитная архитектура



Многослойная архитектура



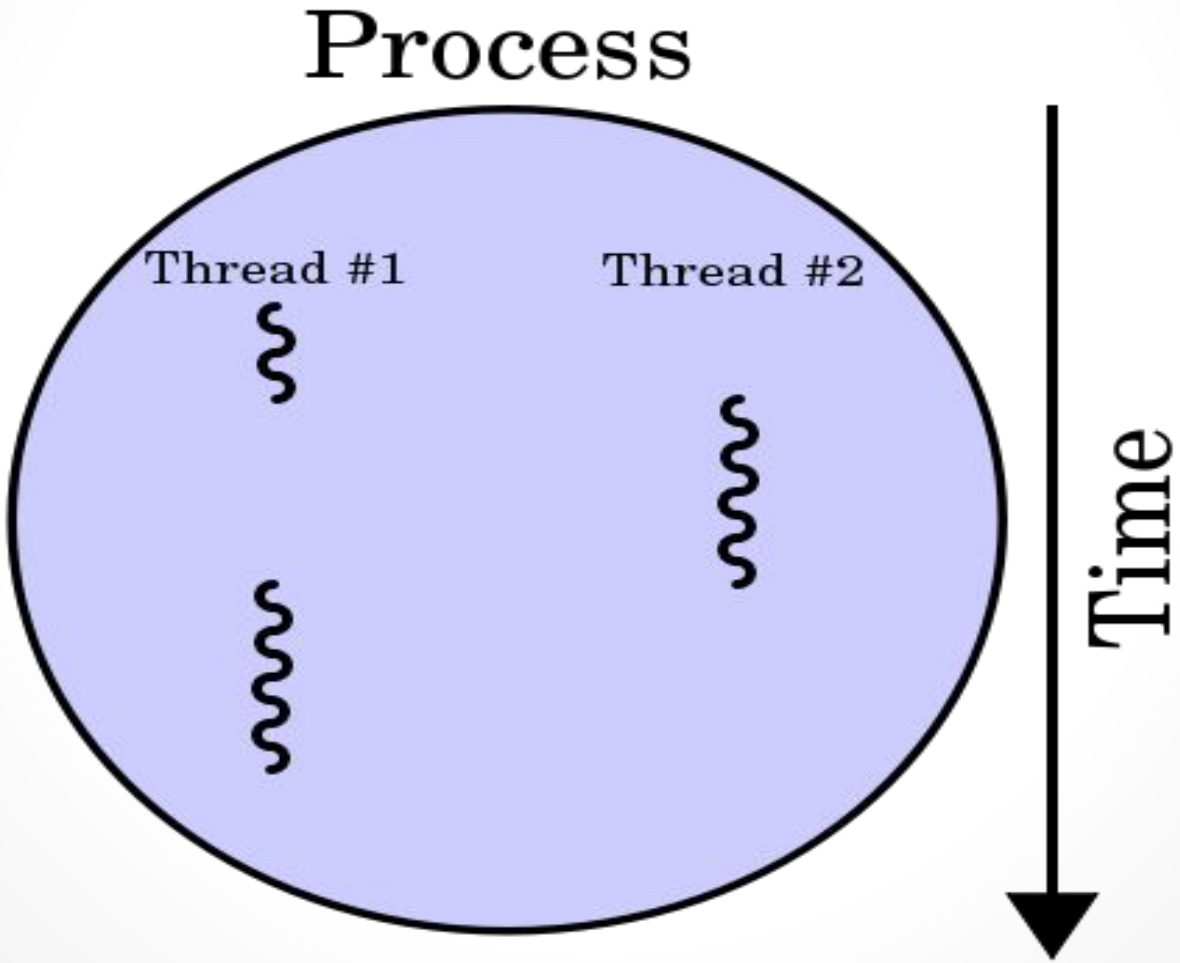
Архитектура «клиент-сервер»



Классификация операционных систем

- **1-й класс:** программирование на уровне микропроцессоров
- **2-й класс:** минимальное ядро системы реального времени.
- **3-й класс:** ядро системы реального времени и инструментальная среда.
- **4-й класс:** ОС с полным сервисом.

Процесс и потоки выполнения



Процесс


старт
приостановлен
готов
готов
исполняется
приостановлен
в ожидании
в ожидании
(заблокирован)

Основные сервисы ОС РВ

- **Управление задачами**. Самая главная группа сервисов. Позволяет разработчикам приложений проектировать программные продукты в виде наборов отдельных программных фрагментов, каждый из которых может относиться к своей тематической области, выполнять отдельную функцию и иметь свой собственный квант времени, отведенный ему для работы. Каждый такой фрагмент называется ***задачей***. Сервисы в рассматриваемой группе обладают способностью запускать задачи и присваивать им приоритеты. Основной сервис здесь — ***планировщик задач***. Он осуществляет контроль за выполнением текущих задач, запускает новые в соответствующий период времени и следит за режимом их работы.
- **Динамическое распределение памяти**. Многие (но не все) ядра ОСРВ поддерживают эту группу сервисов. Она позволяет задачам заимствовать области оперативной памяти для временного использования в работе приложений. Часто эти области впоследствии переходят от задачи к задаче, и посредством этого осуществляется быстрая передача большого количества данных между ними. Некоторые очень малые по размеру ядра ОСРВ, которые предполагается использовать в аппаратных средах со строгим ограничением на объём используемой памяти, не поддерживают сервисы динамического распределения памяти.

Проектирование

Инструменты описания проектов

Диаграммы

- Состояний
- Активности
- Использования
- Компонент
- Размещения
- Последовательностей
- Взаимодействия
- Диаграммы объектов

Диаграммы состояний

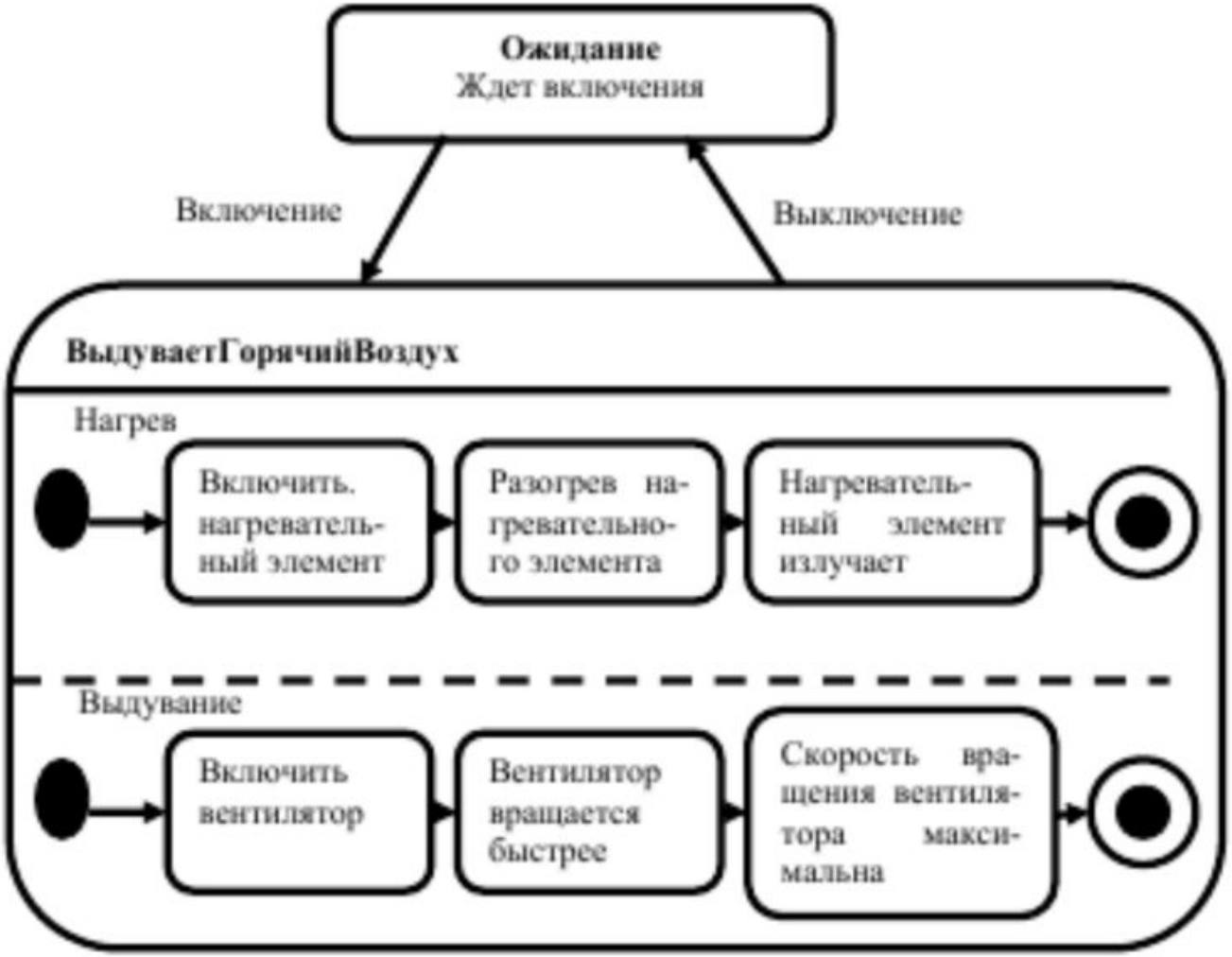
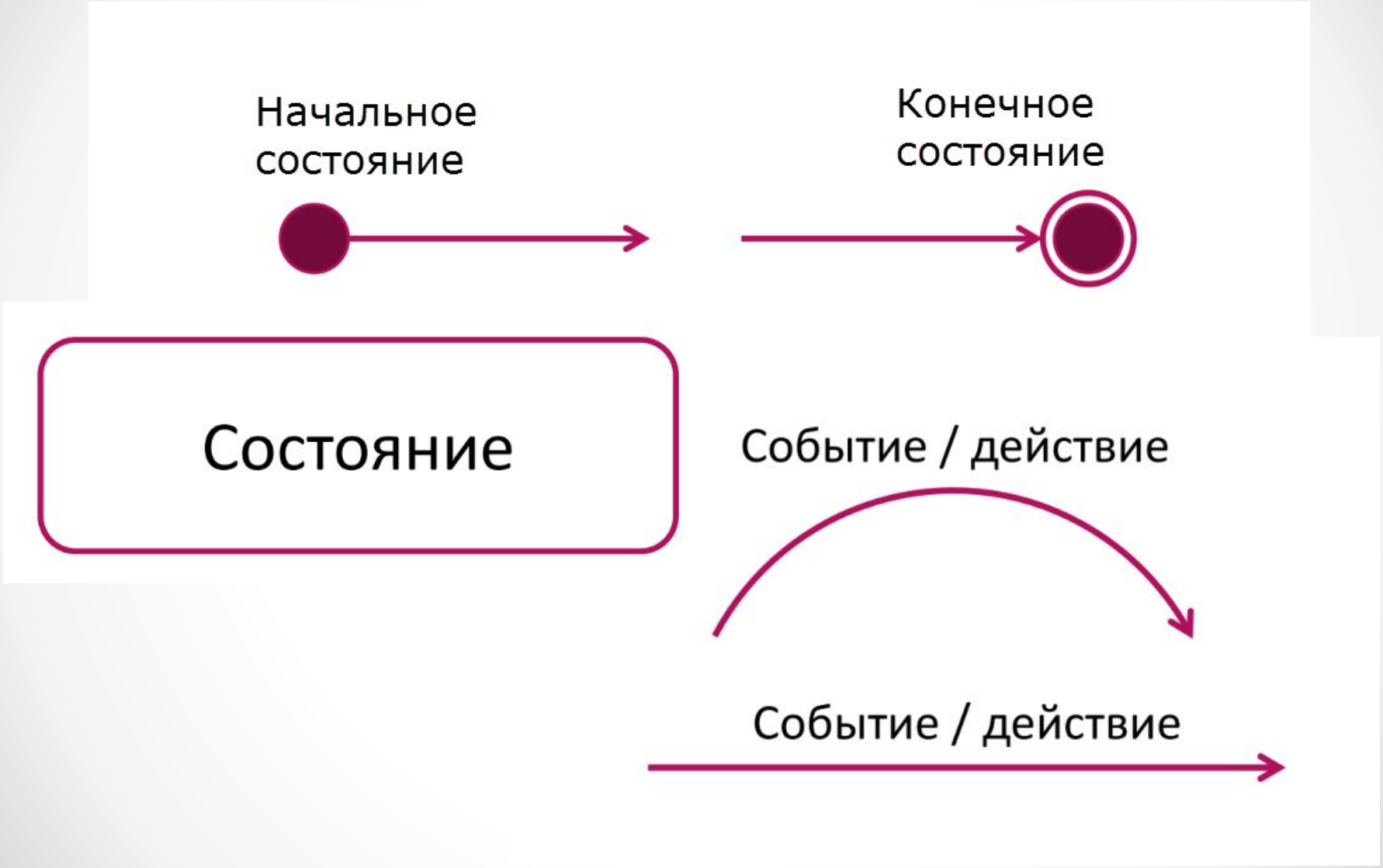


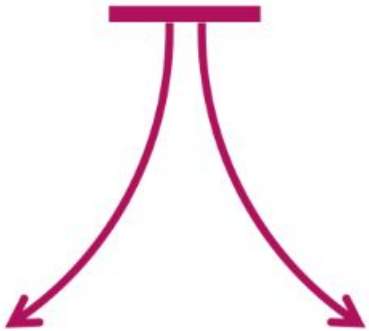
Диаграмма состояний для электросушилки

Диаграммы состояний



Диаграммы состояний

Короткий толстый отрезок с двумя переходами в него показывает ***синхронизацию управления*** . Короткий толстый отрезок с двумя переходами из него представляет ***разветвление потока управления***, что создает множественные состояния.

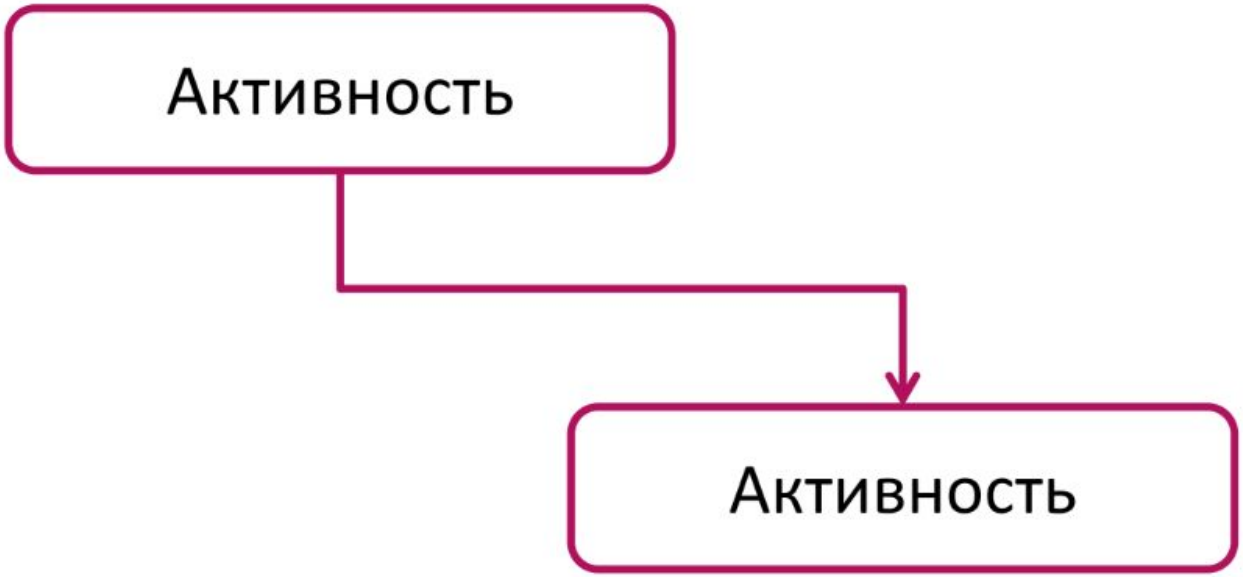


Диаграммы активности

- Частный случай диаграммы состояния.

  Состояния активности отображают непрерываемые действия объектов.

Поток действий показывается связями между состояниями активности.

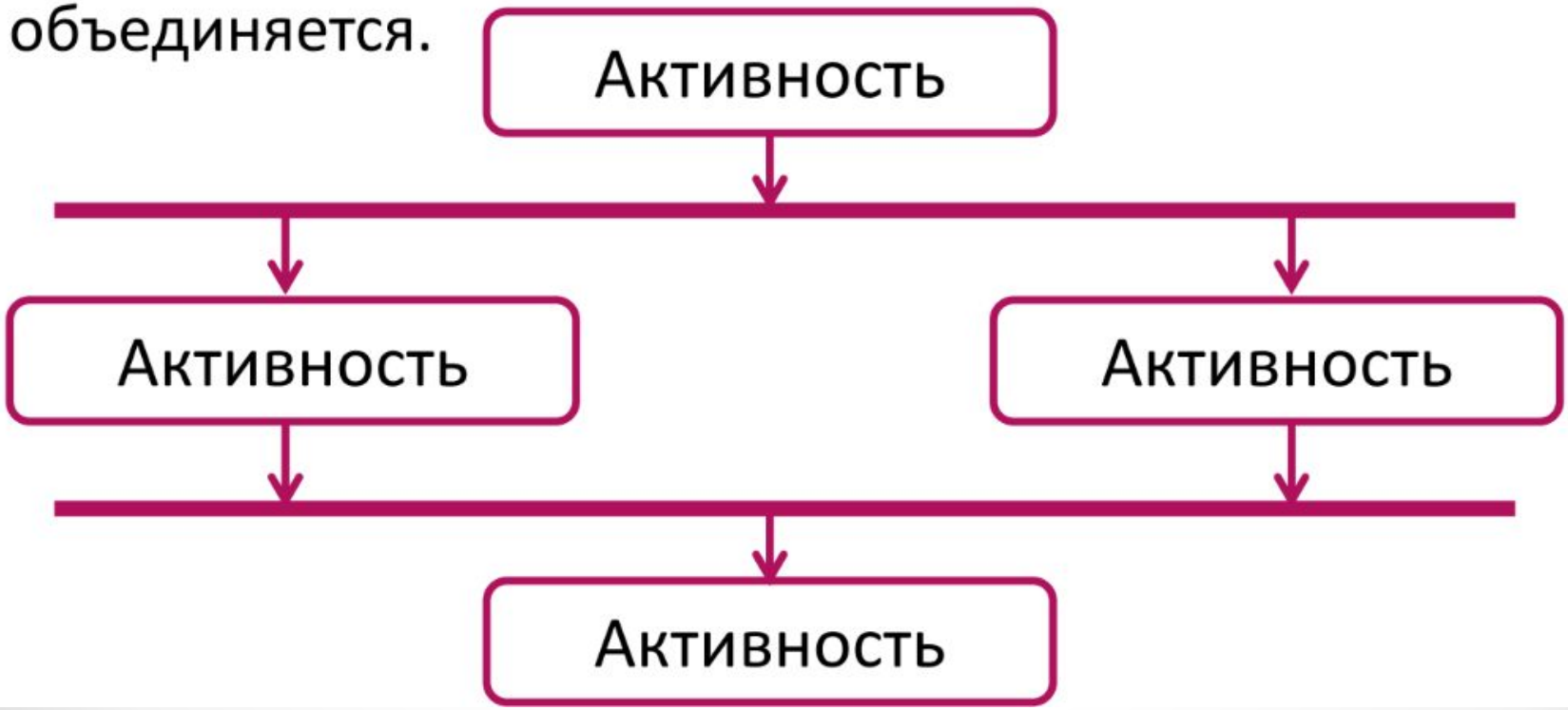


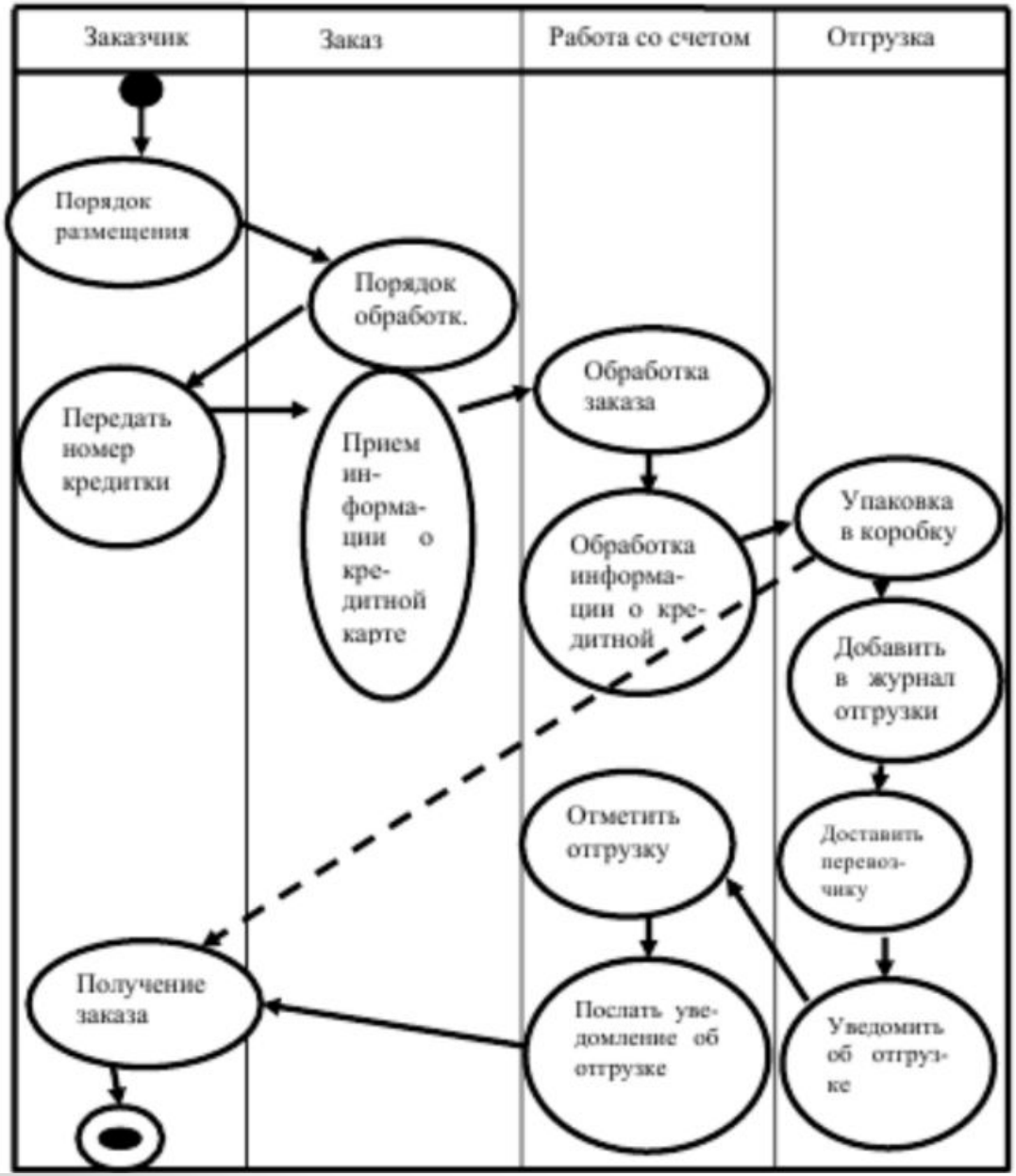
Диаграммы активности

Поток объектов указывает на создание или модификацию объектов активностями. Стрелка от активности к объекту означает, что действие создает или модифицирует объект. Стрелка от объекта к активности означает, что активность использует этот объект.

Диаграммы активности

Линия синхронизации (жирная) помогает описывать параллельные переходы. Синхронизация также называется разветвлением и соединением (forking and joining) - сначала управление разветвляется, затем объединяется.



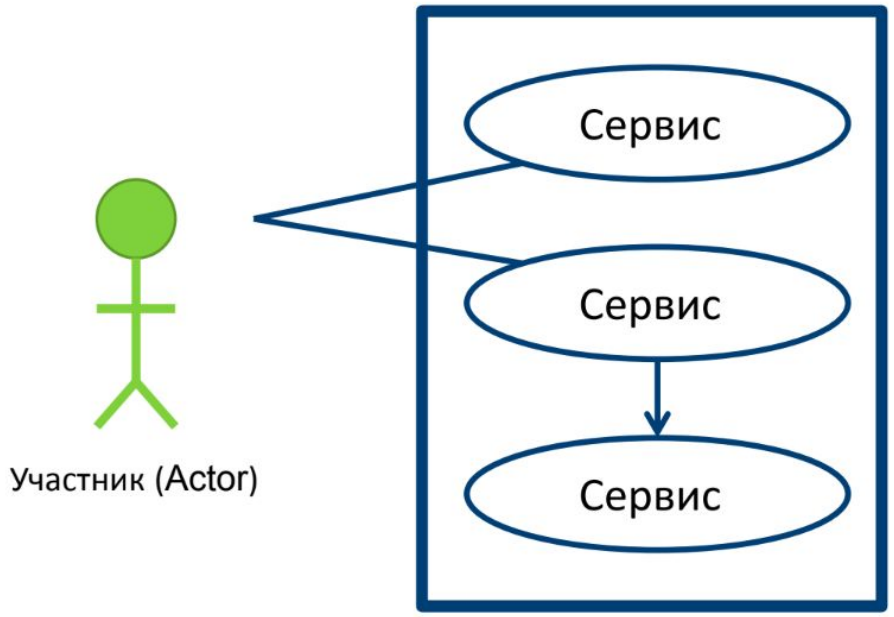


Дорожки объединяют связанные активности в одну колонку. Каждая дорожка помечается ответственным за нее классом.

Диаграмма активности обработки заказов

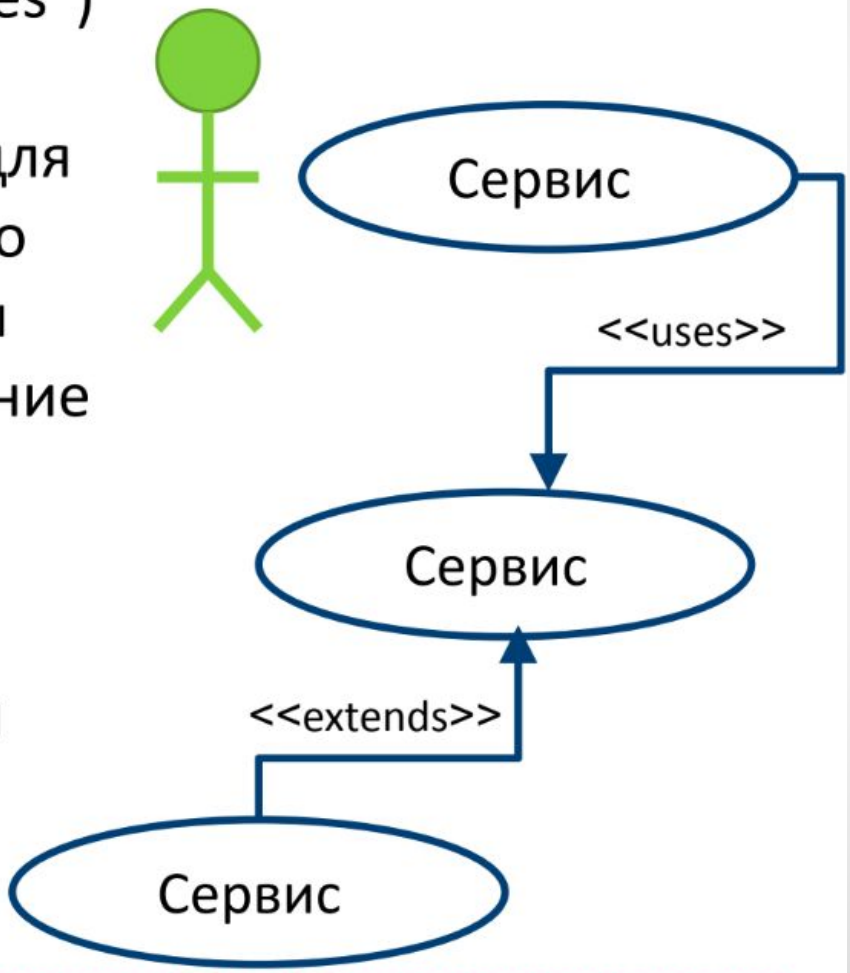
Диаграммы использования

- Моделируют функциональность систем с помощью участников и сервисов.
- Сервисы - это функции, предоставляемые системой пользователям.
- Каждый сервис обозначается овалом. Овал помечается глаголами, дающими представление о назначении системного сервиса.



Диаграммы использования

- Отношение "использует" ("uses") указывает, что один сервис нуждается в услугах другого для выполнения своей задачи. Это необходимо для того, чтобы избежать повторов. Отношение "расширяет" ("extends") указывает альтернативную опцию при определенной ситуации и используется для описания отклонений от нормального поведения.

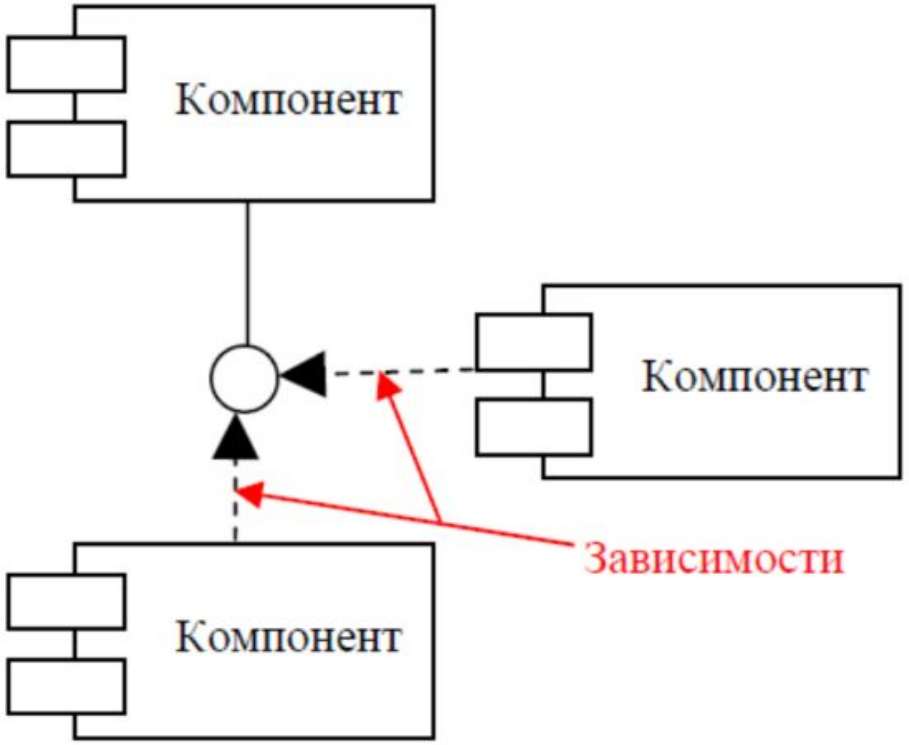


Диаграммы компонент

- Диаграммы компонент описывают организацию физических компонент в системе.
- ***Компонент*** - это физический строительный блок системы. Он представлен прямоугольником с табличками.
- ***Интерфейс*** описывает группу операций используемых или создаваемых компонентами.

Диаграммы компонент

Зависимости отмечаются штриховыми линиями. Они показывают, как изменения в одних компонентах могут повлиять на необходимость изменения других. Учитываются также зависимости по связям и компиляции.

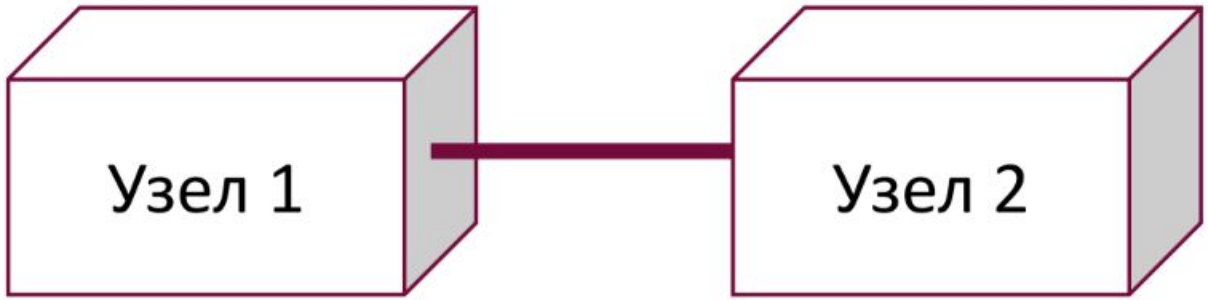


Диаграммы размещений

Диаграммы размещения показывают физические ресурсы системы, включая узлы, компоненты, связи.

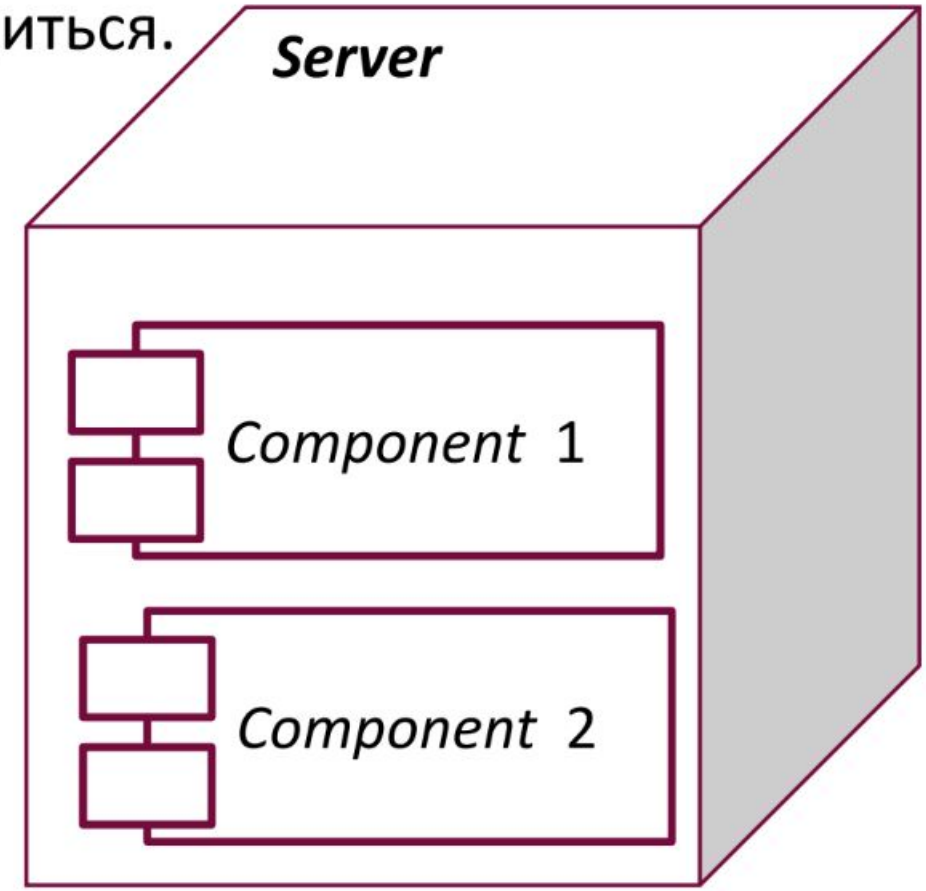
Узел - это физический ресурс, который исполняет коды компонент.

Связь показывает физическое соединение компонент, например, Ethernet, с их помощью представляют пути коммуникаций, по которым система будет передавать информацию.



Диаграммы размещений

Компоненты помещаются внутри узлов там, где они должны находиться.



Размещение компонентов Component в узле Server.

Диаграммы взаимодействия

Диаграммы взаимодействия показывают отношения между объектами в терминах **последовательности** сообщений. Диаграммы взаимодействия представляют комбинацию информации, взятой из диаграмм классов, последовательностей и использования и описывают как статические, так и динамические свойства системы.

- **Роли классов** показывают поведение объектов. Для представления ролей используют символы объектов UML, но без перечисления атрибутов объектов.
- **Роли связей** описывают, как связь будет вести себя в конкретной ситуации.

Для указания цикла используют символ * после номера сообщения.

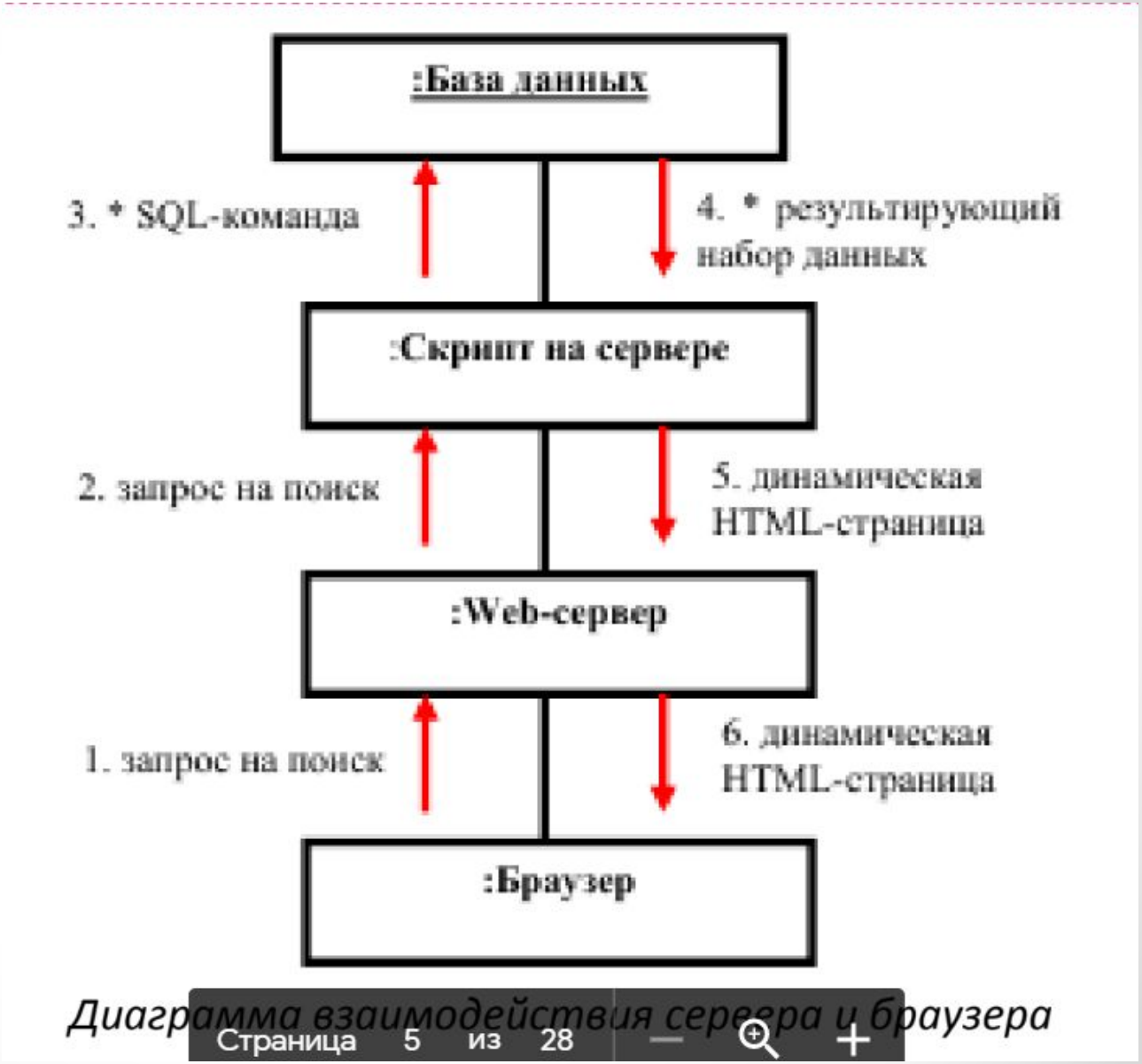
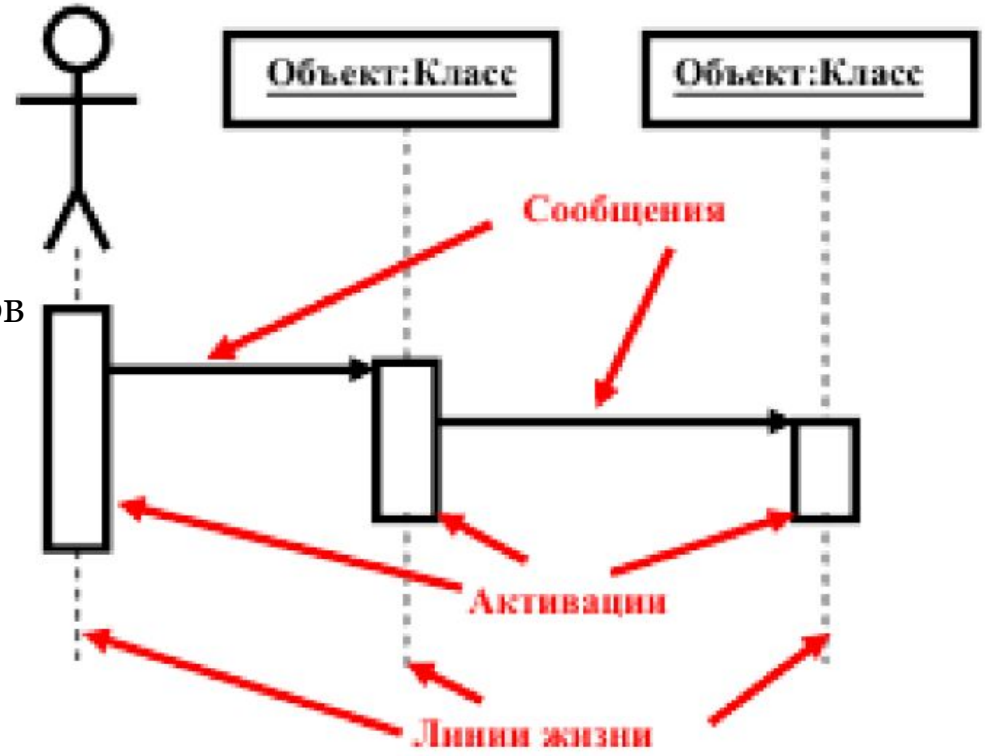


Диаграмма взаимодействия сервера и браузера

Диаграммы последовательностей

Диаграммы последовательностей описывают взаимодействия между классами в терминах обмена сообщениями во времени.

Роли классов описывают поведение объектов.
Используются символы UML-объектов для показа
ролей классов, но без перечисления списка
атрибутов.



Диаграммы последовательностей

Прямоугольники - ***активации*** - представляют время, необходимое объекту для выполнения задания.

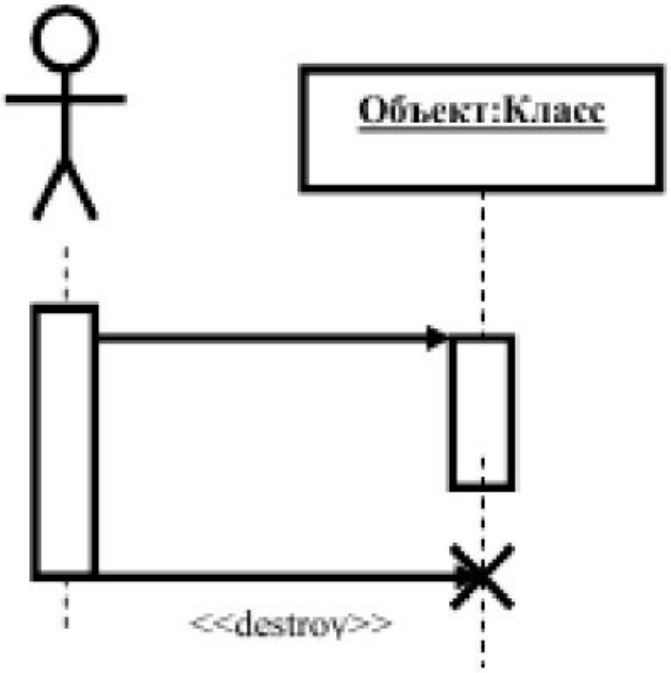
Сообщения обозначаются стрелками трех видов. *Простая стрелка (flat)*  - сообщение не ожидает ответа, управление переходит к получателю (отправитель завершается);

синхронное (call)  - нормальный вызов процедуры - отправитель теряет управление до завершения обработки сообщения получателем, затем управление возвращается отправителю;

асинхронное  - сообщение не требует ответа, но в отличие от случая flat отправитель остается активным и может посылать другие сообщения.

Линии жизни - это вертикальные штриховые линии, которые показывают присутствие объекта во времени.

Объекты могут быть уничтожены с помощью стрелок, помеченных "< < destroy > >" , направленных в X.



Повторение, или ***цикл***, в диаграмме последовательностей показывается прямоугольником. Условие выхода из цикла помещается в его левом нижнем углу в квадратных скобках.

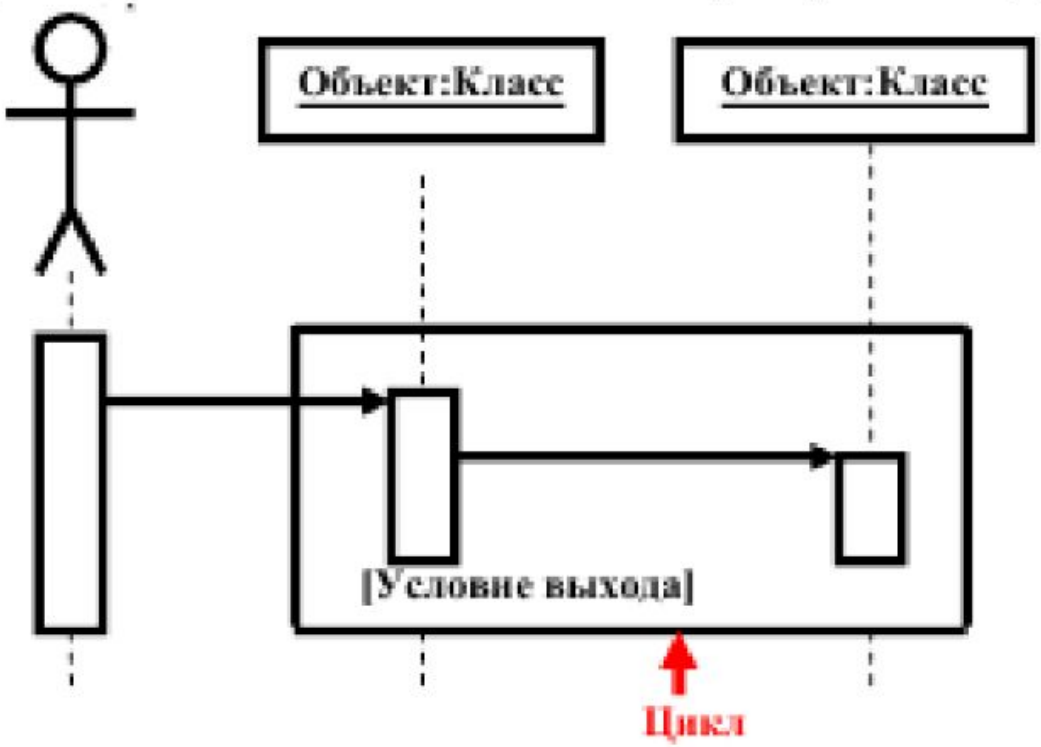
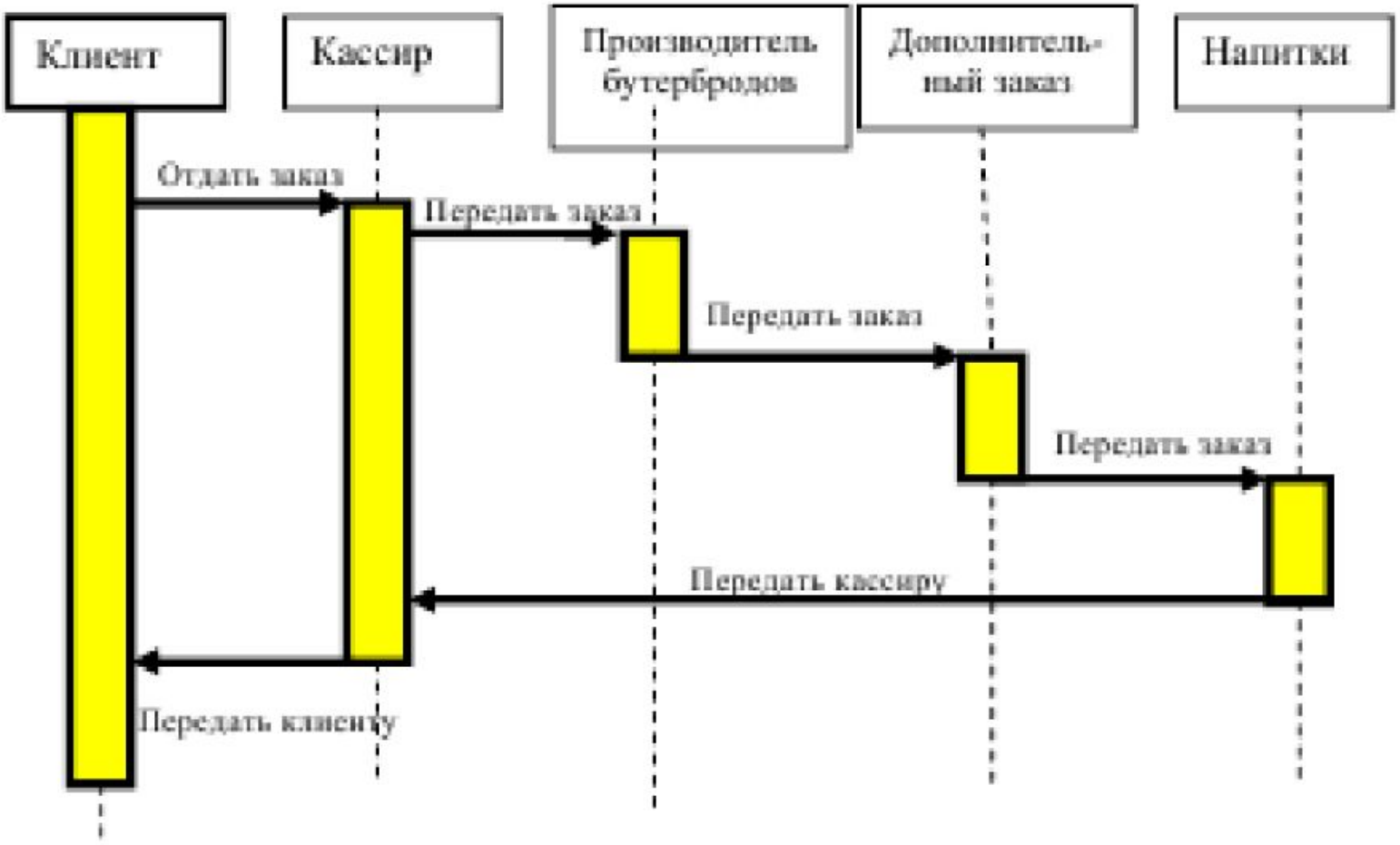
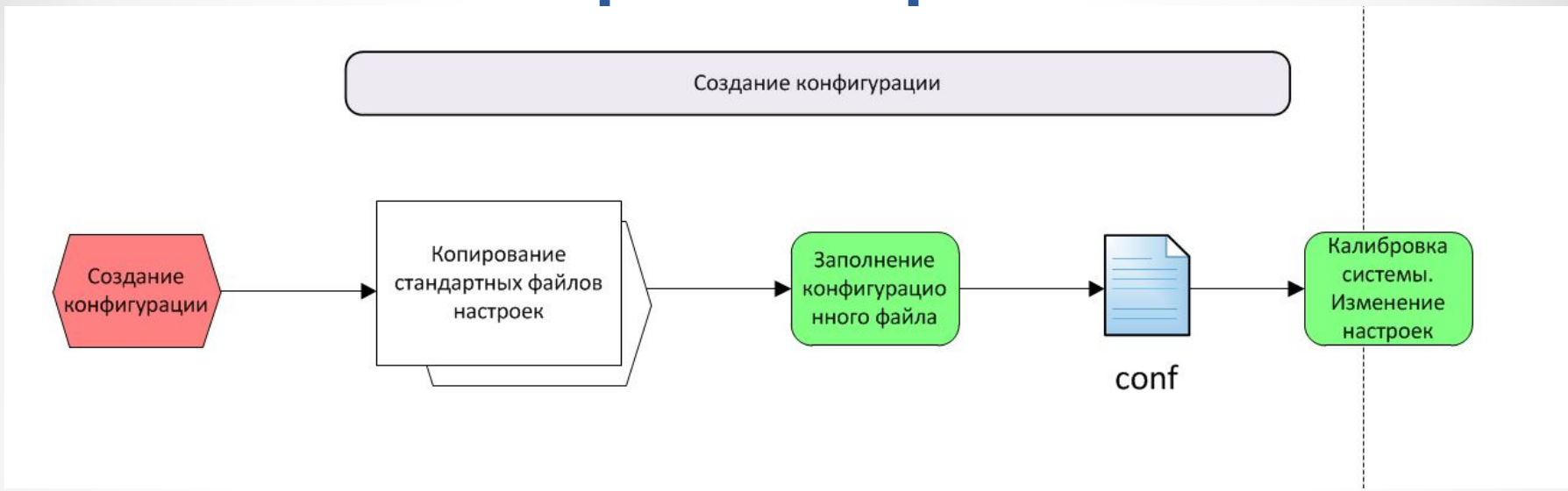
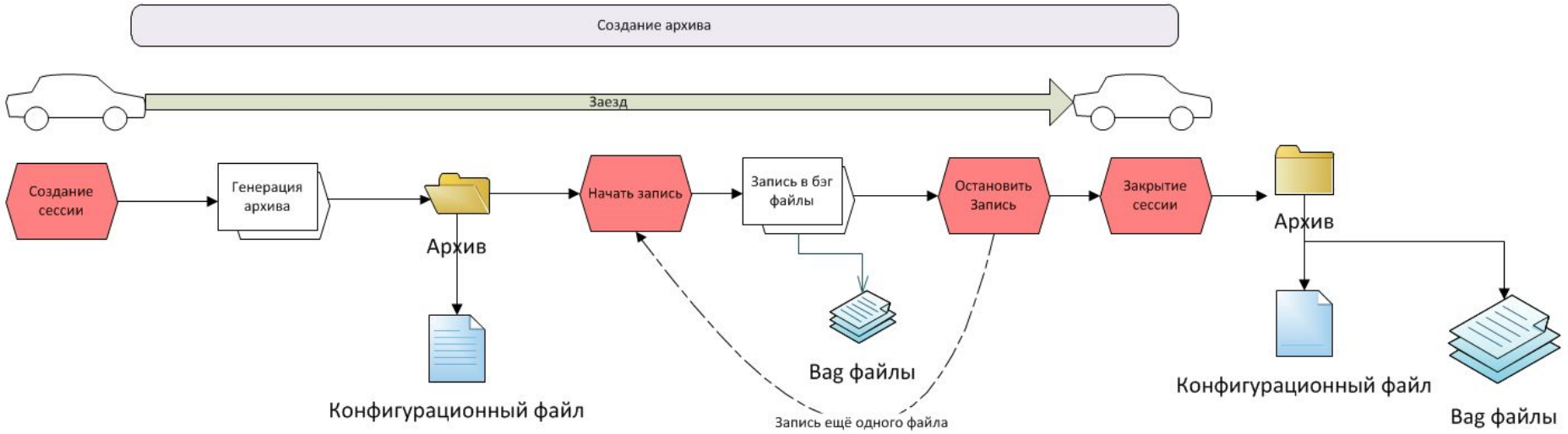


Диаграмма последовательностей для процесса подготовки заказа



Примеры


Создание конфигурации
Создание конфигурации
Копирование стандартных файлов настроек
Заполнение конфигурационного файла
conf
Калибровка системы. Изменение настроек


Создание архива
Заезд
Создание сессии
Генерация архива
Архив
Конфигурационный файл
Начать запись
Запись в бэг файлы
Bag файлы
Остановить Запись
Запись ещё одного файла
Закрытие сессии
Архив
Конфигурационный файл
Bag файлы

Диаграммы объектов

Стандарт UML

Объект

- Объект (object) - экземпляр класса.
- Также про объект можно сказать, что он уникально идентифицируется значениями атрибутов, определяющими его состояние в данный момент времени.

- Диаграммы показывают множество объектов - экземпляров классов и отношений между ними в некоторый момент времени. То есть диаграмма объектов – это своего рода снимок состояния системы в определенный момент времени, показывающий множество объектов, их состояния и отношения между ними в данный момент.

UML - Unified Modeling Language

Зависимость

- Обозначает такое отношение между классами, что изменение спецификации класса-поставщика может повлиять на работу зависимого класса, но не наоборот.

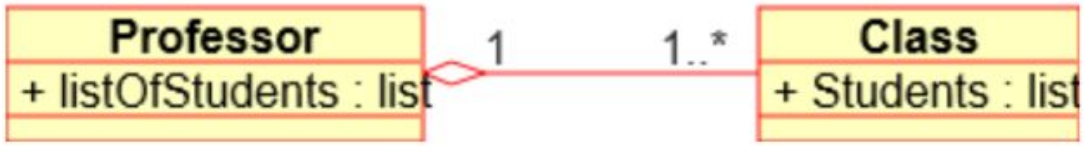
Ассоциация

- Показывает, что объекты одной сущности (класса) связаны с объектами другой сущности таким образом, что можно перемещаться от объектов одного класса к другому. Является общим случаем композиции и агрегации.
- Например, класс Человек и класс Школа имеют ассоциацию, так как человек может учиться в школе. Ассоциации можно присвоить имя «учится в».

UML

Агрегация

- это разновидность ассоциации при отношении между целым и его частями. Как тип ассоциации агрегация может быть именованной. Одно отношение агрегации не может включать более двух классов (контейнер и содержимое).
- *Агрегация* встречается, когда один класс является коллекцией или контейнером других. Причём по умолчанию, агрегацией называют *агрегацию по ссылке*, то есть когда время существования содержащихся классов не зависит от времени существования содержащего их класса. Если контейнер будет уничтожен, то его содержимое — нет.
- Графически агрегация представляется пустым ромбом на блоке класса, и линией, идущей от этого ромба к содержащемуся классу.



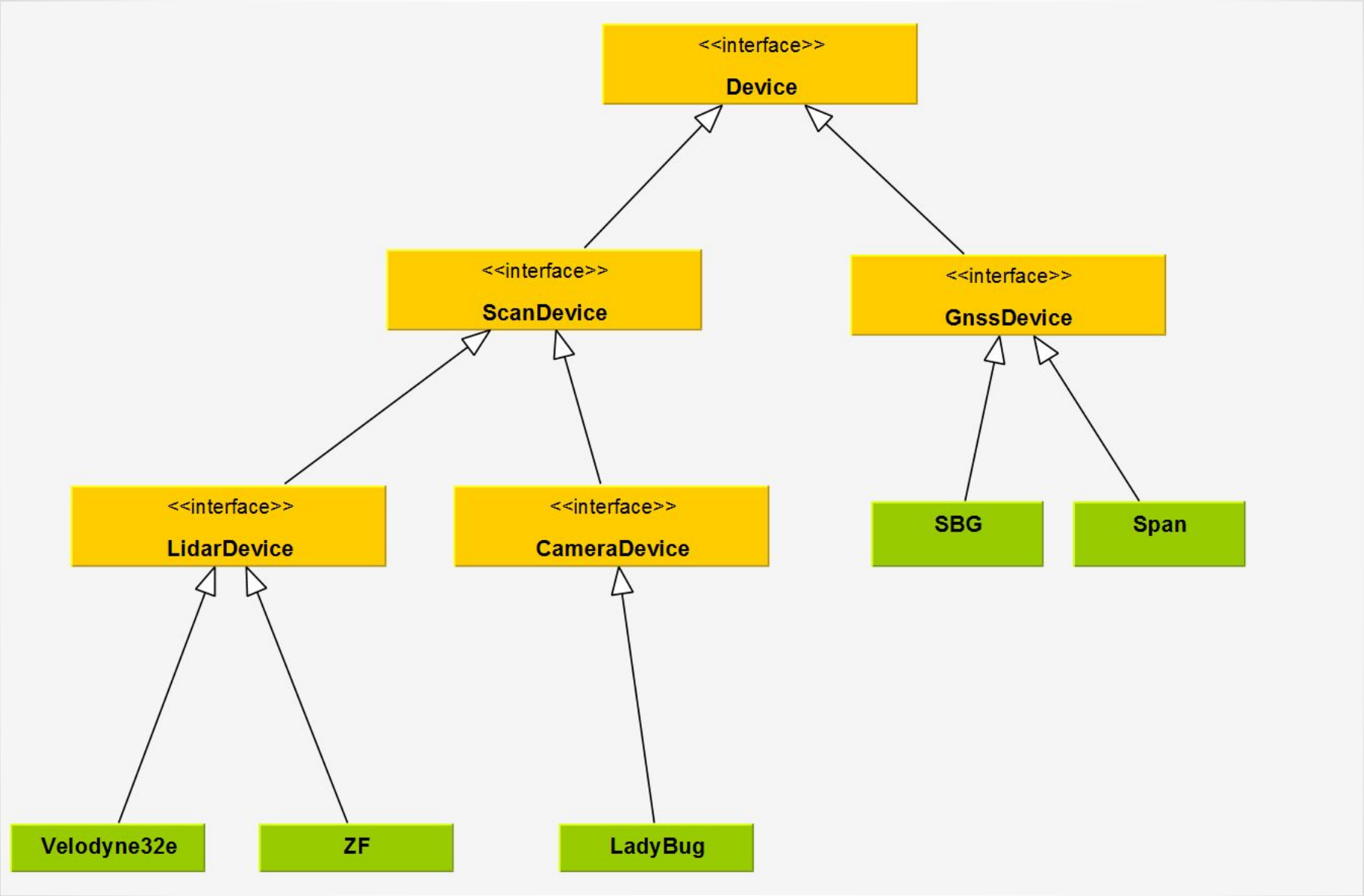
UML

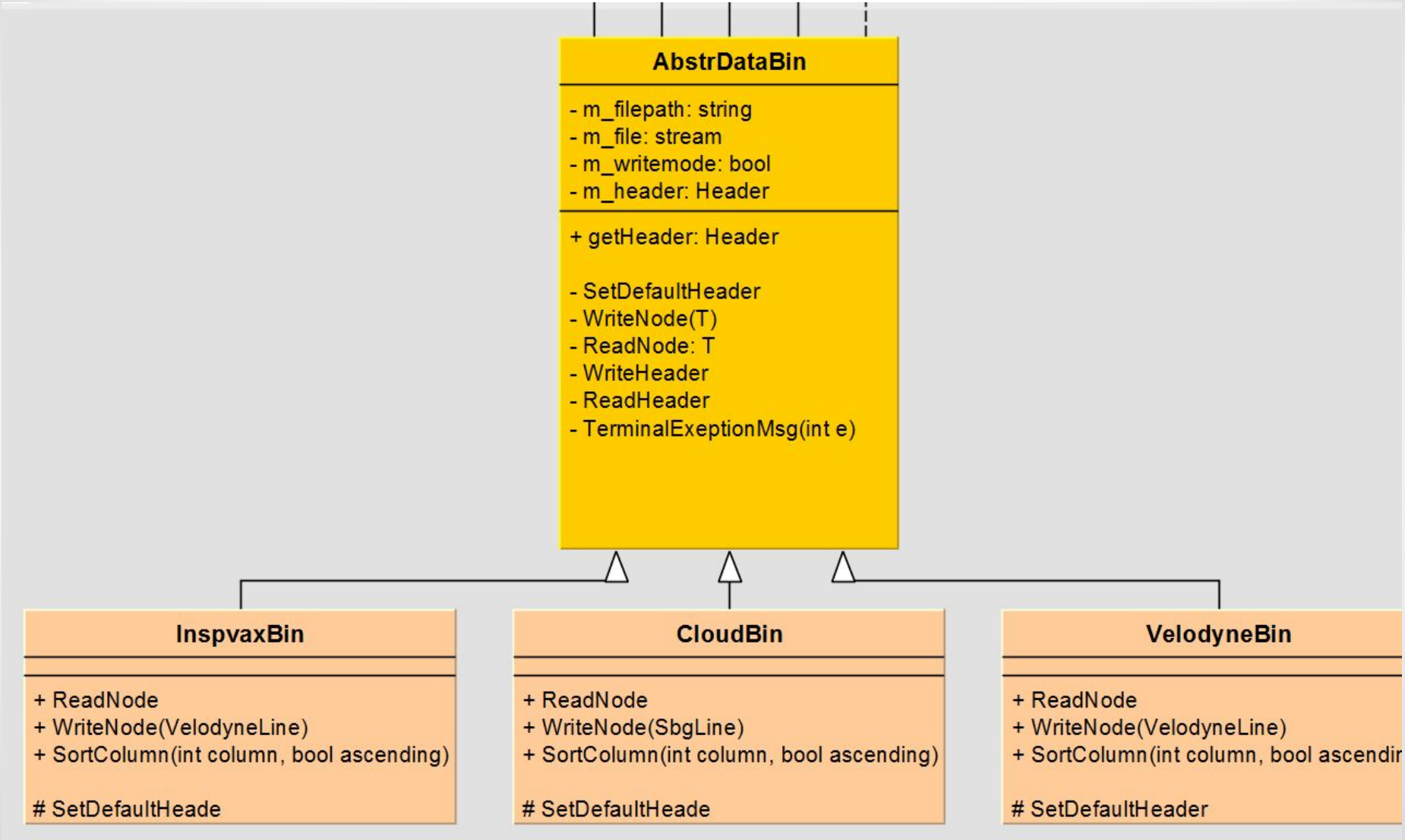
Композиция

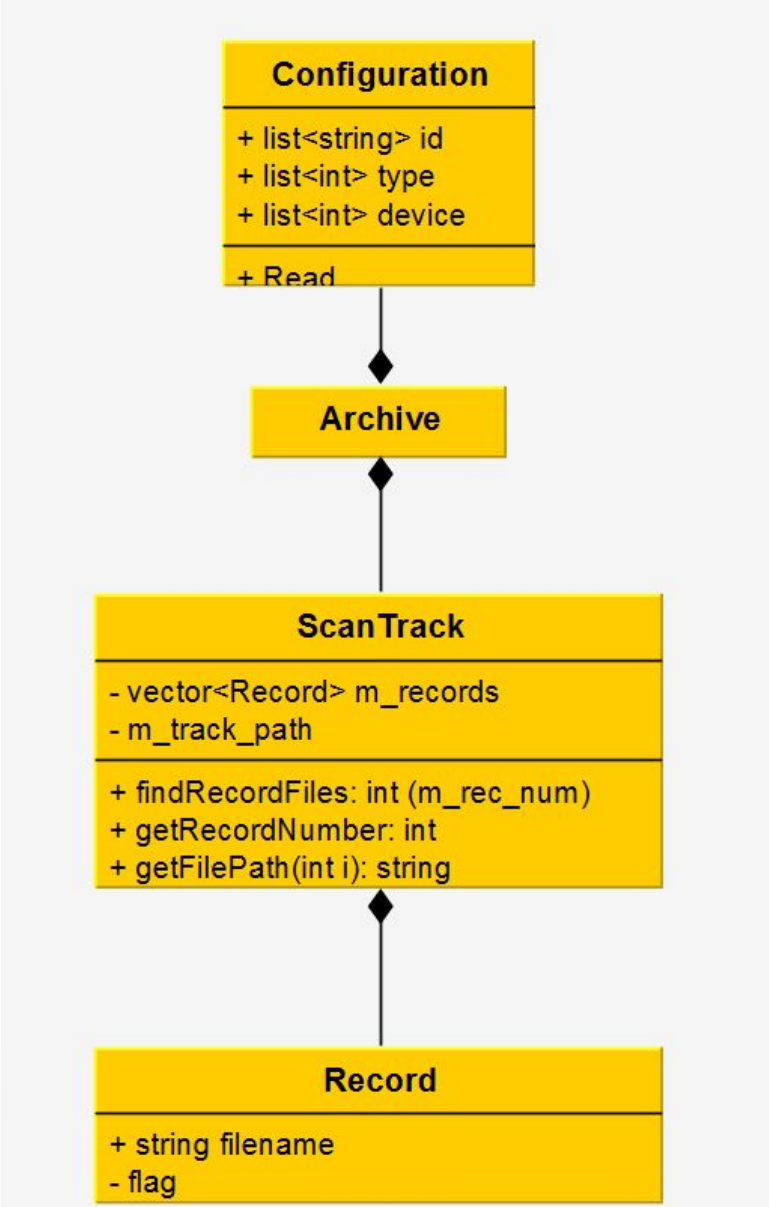
- Более строгий вариант агрегации. Известна также как агрегация по значению.
- *Композиция* имеет жёсткую зависимость времени существования экземпляров класса контейнера и экземпляров содержащихся классов. Если контейнер будет уничтожен, то всё его содержимое будет также уничтожено.
- Графически представляется как и агрегация, но с закрашенным ромбиком.

Различия между композицией и агрегацией

- Приведём наглядный пример. Комната является частью квартиры, следовательно здесь подходит композиция, потому что комната без квартиры существовать не может. А, например, мебель не является неотъемлемой частью квартиры, но в то же время, квартира содержит мебель, поэтому следует использовать агрегацию.


<<interface>>
Device
<<interface>>
ScanDevice
<<interface>>
GnssDevice
<<interface>>
LidarDevice
<<interface>>
CameraDevice
SBG
Span
Velodyne32e
ZF
LadyBug


AbstrDataBin
- m_filepath: string
- m_file: stream
- m_writemode: bool
- m_header: Header
+ getHeader: Header
- SetDefaultHeader
- WriteNode(T)
- ReadNode: T
- WriteHeader
- ReadHeader
- TerminalExeptionMsg(int e)
InspvaxBin
+ ReadNode
+ WriteNode(VelodyneLine)
+ SortColumn(int column, bool ascending)
# SetDefaultHeade
CloudBin
+ ReadNode
+ WriteNode(SbgLine)
+ SortColumn(int column, bool ascending)
# SetDefaultHeade
VelodyneBin
+ ReadNode
+ WriteNode(VelodyneLine)
+ SortColumn(int column, bool ascendir
# SetDefaultHeader


Configuration
+ list<string> id
+ list<int> type
+ list<int> device
+ Read
Archive
ScanTrack
- vector<Record> m_records
- m_track_path
+ findRecordFiles: int (m_rec_num)
+ getRecordNumber: int
+ getFilePath(int i): string
Record
+ string filename
- flag