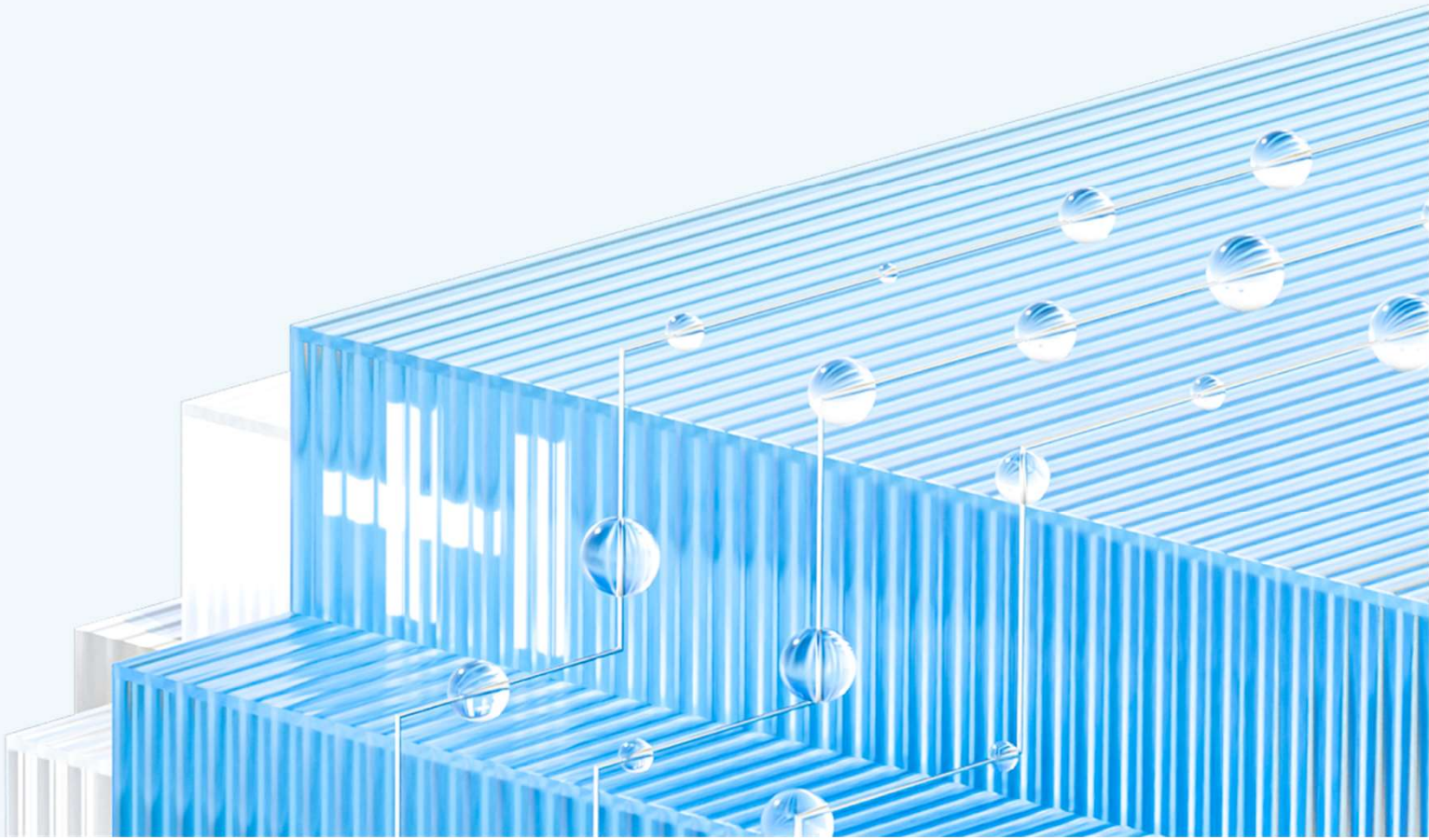
# Основы языка UML



+I TI

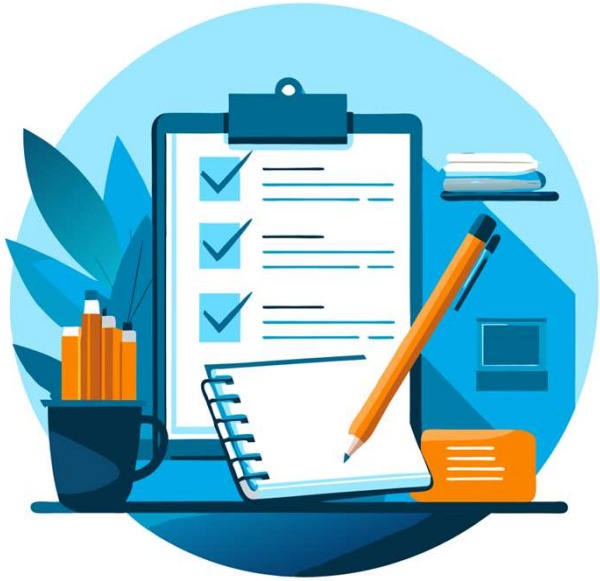
# Содержание

1. 14 видов диаграмм: зачем, как выглядят, как читать
2. Основные диаграммы для всех ролей (системный аналитик, архитектор ИС, инженер MES)
3. Концептуальные области UML
4. Место UML в процессе разработки ПО

# 1. 14 видов диаграмм: +I
# зачем, как выглядят, как читать

# Что такое UML?

**UML** — язык графического моделирования для визуализации, проектирования и документирования систем.



**Категории диаграмм:**

+ структурные (статическая структура системы);
+ поведенческие (динамика и процессы);
+ диаграммы взаимодействия (подвид поведенческих).

# Структурные диаграммы (Static Structure Diagrams)

**Что это**

Диаграммы, которые показывают статическую структуру системы:
её компоненты, объекты, классы, их атрибуты и связи.
Они не зависят от времени и описывают скелет системы.

**Нужны, чтобы**:

+ моделировать архитектуру системы;
+ определять, из каких сущностей состоит система и как они связаны;
+ документировать данные, сервисы и инфраструктуру.

**Ключевые моменты**:

+ показывают что существует в системе, а не как это работает;
+ примеры: классы, таблицы БД, серверы, модули.

# Структурные диаграммы (Static Structure Diagrams)

- **Диаграмма классов (Class Diagram)** показывает классы, их атрибуты, методы и связи (ассоциации, наследование, агрегацию).
- **Диаграмма компонентов (Component Diagram)** отображает физические компоненты системы (модули, библиотеки, сервисы).
- **Диаграмма пакетов (Package Diagram)** отображает физические компоненты системы (модули, библиотеки, сервисы).
- **Диаграмма развёртывания (Deployment Diagram)** показывает, как компоненты распределены по железу (серверы, сети, филиалы).
- **Диаграмма композитной структуры (Composite Structure Diagram)** демонстрирует внутреннюю структуру сложных объектов.
- **Диаграмма объектов (Object Diagram) —** снимок системы в конкретный момент времени (экземпляры классов).
- **Диаграмма профилей (Profile Diagram)** расширяет UML для конкретной предметной области.

# Поведенческие диаграммы (Behavioral Diagrams)

**Что это**

Описывают динамические аспекты системы: процессы, сценарии взаимодействия, изменения состояний объектов.
Они отвечают на вопрос «Как система работает?».

**Нужны, чтобы**:

+ описывать бизнес-процессы и алгоритмы;
+ моделировать реакции системы на события;
+ документировать требования пользователей.

**Ключевые моменты**:

+ показывают поведение системы во времени;
+ примеры: процессы одобрения кредита, жизненный цикл карты.

# Поведенческие диаграммы (Behavioral Diagrams)

- **Диаграмма вариантов использования (Use Case Diagram)** показывает взаимодействие акторов (пользователей, систем) с функционалом.
- **Диаграмма активностей (Activity Diagram)** визуализирует бизнес-процессы и алгоритмы (потоки, решения, параллельные действия).
- **Диаграмма состояний (State Machine Diagram)** отслеживает изменение состояний объекта в ответ на события.

**Зачем используются в банке**
Поведенческие диаграммы незаменимы для проектирования процессов, соответствующих регуляторным требованиям, и оптимизации клиентских сценариев (например, мобильного банкинга).

# Диаграммы взаимодействия (Interaction Diagrams)

**Что это**

Подвид поведенческих диаграмм, который фокусируется на обмене сообщениями между объектами в рамках конкретного сценария. Показывают, как объекты координируются для достижения цели.

**Нужны, чтобы**:

+ детализировать сложные процессы с участием нескольких компонентов;
+ выявлять ошибки в последовательности действий;
+ оптимизировать взаимодействие между модулями.

**Ключевые моменты**:

+ акцент на порядок и условия передачи сообщений;
+ примеры: авторизация платежа, обработка запроса в API.

# Диаграммы взаимодействия (Interaction Diagrams)

- **Диаграмма последовательности (Sequence Diagram)** показывает пошаговое взаимодействие объектов во времени.
- **Диаграмма коммуникации (Communication Diagram)** делает акцент на связи между объектами, а не на времени (альтернатива Sequence Diagram).
- **Диаграмма временно́й синхронизации (Timing Diagram)** учитывает временны́е ограничения в процессах.
- **Диаграмма обзора взаимодействия (Interaction Overview Diagram)** комбинирует несколько диаграмм взаимодействия в высокоуровневый процесс.

**Зачем используется в банке**

Диаграммы взаимодействия помогают выявить узкие места в процессах (например, задержки в коммуникации с эквайрингом), спроектировать отказоустойчивые сценарии и согласовать логику между командами разработки и бизнесом.

# Сравнение категорий диаграмм

| Категория | Фокус | Примеры в банках |
|---|---|---|
| **Структурные** | Что есть в системе? | Счета, клиенты, серверы, модули |
| **Поведенческие** | Как система работает? | Процесс одобрения кредита, статусы карт |
| **Взаимодействия** | Как объекты общаются? | Цепочка запросов при переводе средств |

# 2. Основные диаграммы для всех ролей

**Системный аналитик, архитектор ИС, инженер MES**

# Диаграмма классов (структурная)

## Зачем

- Моделирует структуру данных системы (классы, атрибуты, методы, связи).

## Как выглядит

- Классы изображаются прямоугольниками с разделами: имя, атрибуты, методы.

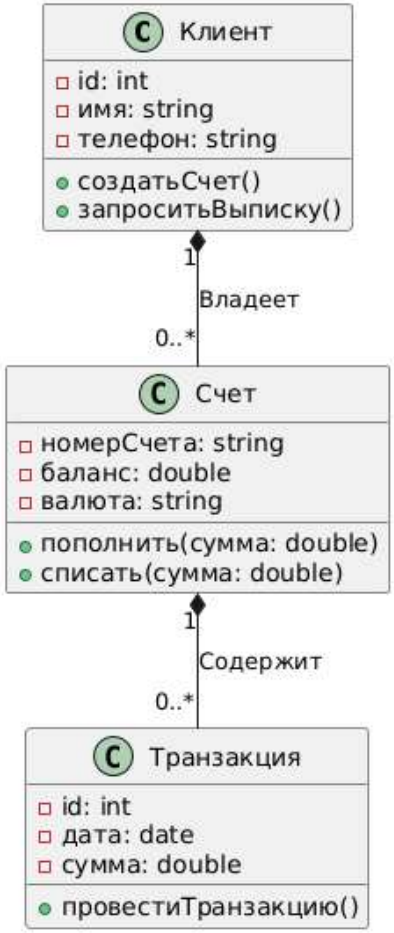
## Связи

- Наследование (стрелка с треугольником).
- Ассоциации (линии).
- Агрегация/композиция (ромбы).

## Как читать

- Определите ключевые сущности и их взаимодействие.

# Диаграмма классов




```
@startuml
class Клиент {
  - id: int
  - имя: string
  - телефон: string
  + создатьСчёт()
  + запроситьВыписку()
}

class Счёт {
  - номерСчёта: string
  - баланс: double
  - валюта: string
  + пополнить(сумма: double)
  + списать(сумма: double)
}

class Транзакция {
  - id: int
  - дата: date
  - сумма: double
  + провестиТранзакцию()
```

# Диаграмма компонентов (структурная)

## Зачем

- Показать, из каких компонентов состоит система (модули, сервисы).

## Как выглядит

- Прямоугольники с иконкой компонента и интерфейсы (маленькие круги).

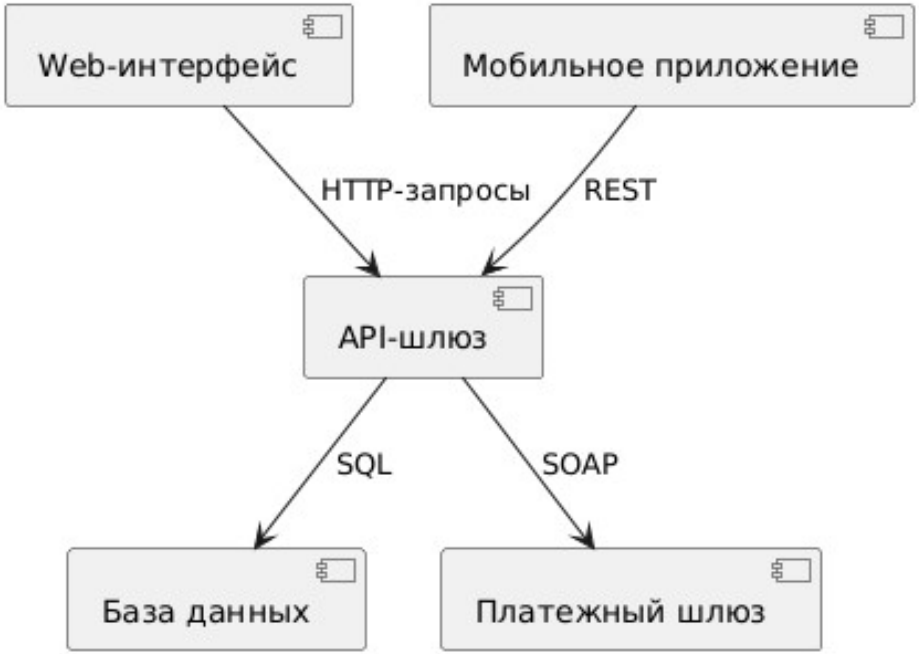
## Ошибки

- Не учитывать зависимости между компонентами (например, мобильное приложение зависит от API банка).

## Как читать

- Ищите, какие компоненты взаимодействуют через интерфейсы.

# Диаграмма компонентов




```
@startuml
component "Веб-интерфейс" as Web
component "Мобильное приложение" as
Mobile
component "API-шлюз" as API
component "База данных" as DB
component "Платёжный шлюз" as Payment

Web --> API : HTTP-запросы
Mobile --> API : REST
API --> DB : SQL
API --> Payment : SOAP
@enduml
```

# Диаграмма объектов (структурная)

## Зачем

- Показать конкретные экземпляры классов в определённый момент времени.

## Как выглядит

- Объекты (прямоугольники с подчёркнутыми именами) и их связи.

## Ошибки

- Использовать вместо диаграммы классов (объекты — это экземпляры, а не шаблоны).

## Как читать

- Смотрите на конкретные данные, а не абстрактные классы.

# Диаграмма объектов


```
@startuml

!define BankObjectDiagram

' Объекты (экземпляры классов)
object "Иван Петров: Клиент" as client {
  id = 12345
  имя = "Иван Петров"
  телефон = "+7 999 123-45-67"
}

object "ACC-2023-RUB: Счёт" as account {
  номер = "ACC-2023-RUB"
  баланс = 50 000.00 ₽
  валюта = "RUB"
}

object "TXN-001: Транзакция" as tx1 {
  id = "TXN-001"
  сумма = +10 000.00 ₽
  дата = "2025-05-20"
  описание = "Пополнение через терминал"
}

object "TXN-002: Транзакция" as tx2 {
  id = "TXN-002"
  сумма = -5 000.00 ₽
  дата = "2025-05-19"
  описание = "Оплата услуг ЖКХ"
}

' Связи между объектами
client --> account : владеет
account --> tx1 : содержит
account --> tx2 : содержит

note right of account
  **Текущее состояние счёта:**
  - Дата: 2025-05-21
  - Доступный остаток: 50 000 ₽
end note

@enduml
```

# Диаграмма пакетов (структурная)

## Зачем

- Группировать связанные элементы системы (классы, компоненты) в пакеты.

## Как выглядит

- Папки с названиями пакетов и зависимости между ними (пунктирные стрелки).

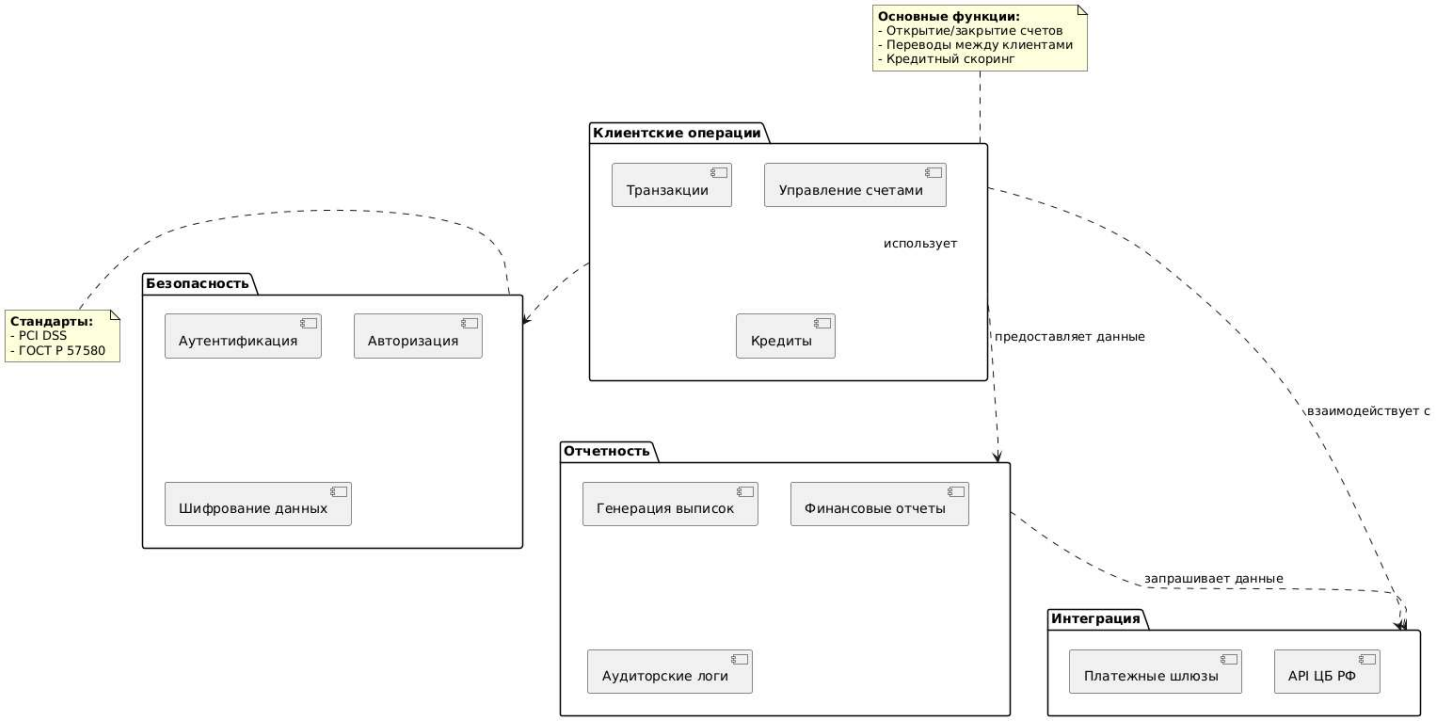
## Ошибки

- Создавать слишком крупные пакеты без чёткой логики.

## Как читать

- Определите, как пакеты взаимодействуют и какие модули связаны.

# Диаграмма пакетов




```
@startuml
 Пакеты (модули системы)
package "Клиентские операции" as client_ops {
 [Управление счетами]
 [Транзакции]
 [Кредиты]}
package "Безопасность" as security {
 [Аутентификация]
 [Авторизация]
 [Шифрование данных]}
package "Отчётность" as reports {
 [Генерация выписок]
 [Финансовые отчёты]
 [Аудиторские логи]}
package "Интеграция" as integration {
 [Платёжные шлюзы]
 [API ЦБ РФ]}
' Зависимости между пакетами
client_ops ..> security : использует
client_ops ..> reports : предоставляет данные
client_ops ..> integration : взаимодействует с
reports ..> integration : запрашивает данные
' Заметки для пояснения
note top of client_ops
 **Основные функции:**
 - открытие/закрытие счетов
 - переводы между клиентами
 - кредитный скоринг
end note
note left of security
 **Стандарты:**
 - PCI DSS
 - ГОСТ Р 57580
end note
@enduml
```

# Диаграмма развёртывания (структурная)

## Зачем

- Показать физическую инфраструктуру системы (серверы, устройства, сеть).

## Как выглядит

- Узлы (кубы) с компонентами внутри и связи между ними.

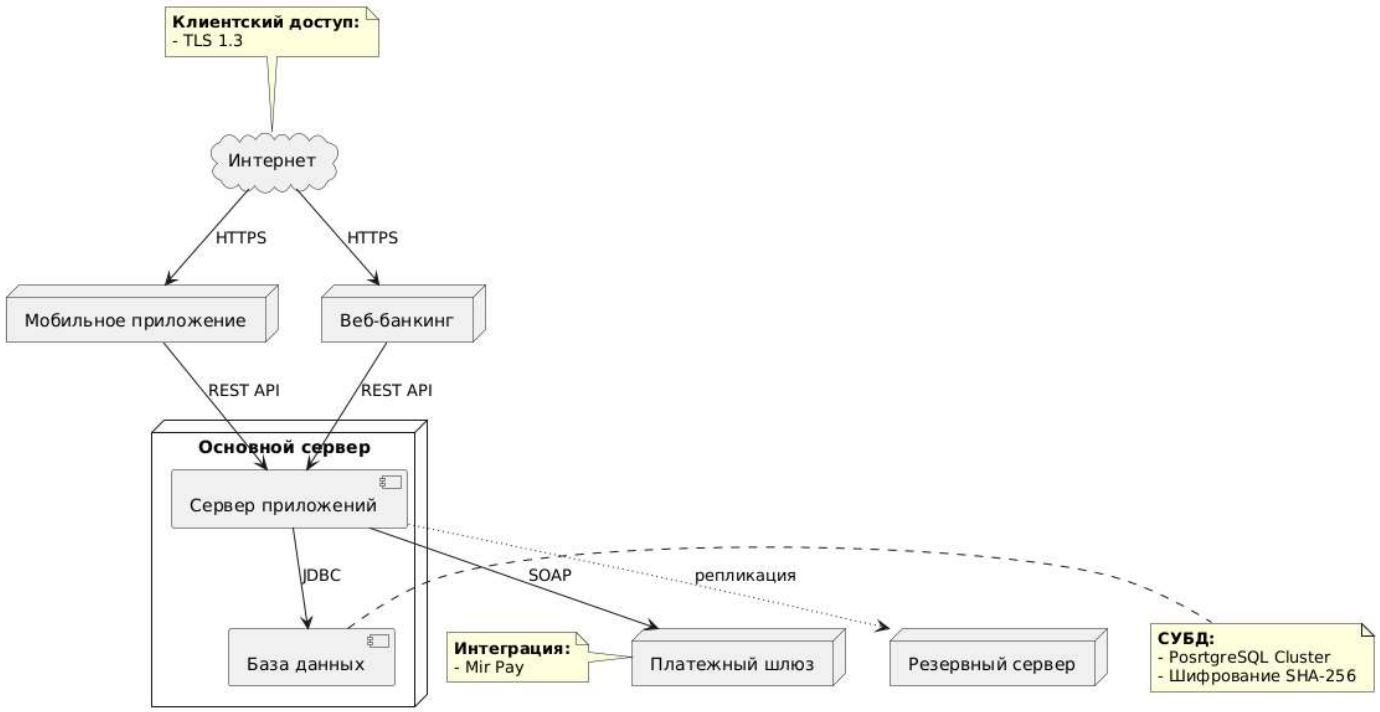
## Ошибки

- Игнорировать сетевую безопасность (например, незашифрованные соединения).

## Как читать

- Определите, где размещены компоненты и как они соединены.

# Диаграмма развёртывания




```
@startuml
 Узлы (физические/виртуальные устройства)
node "Мобильное приложение" as mobile
node "Веб-банкинг" as web
cloud "Интернет" as internet

node "Основной сервер" as server {
  [Сервер приложений] as app_server
  [База данных] as db
}
node "Платёжный шлюз" as payment_gateway
node "Резервный сервер" as backup_server
' Связи между узлами
internet --> mobile : HTTPS
internet --> web : HTTPS
web --> app_server : REST API
mobile --> app_server : REST API
app_server --> db : JDBC
app_server --> payment_gateway : SOAP
app_server -[dotted]-> backup_server : репликация
' Заметки для пояснения
note top of internet
  **Клиентский доступ:**
  - TLS 1.3
end note
note right of db
  **СУБД:**
  - PosrtgreSQL Cluster
  - шифрование SHA-256
end note

note left of payment_gateway
  **Интеграция:**
  - Mir Pay
end note

@enduml
```

# Диаграмма активностей (поведенческая)

## Зачем

- Описать бизнес-процессы или алгоритмы (шаги, решения, параллельные действия).

## Как выглядит

- Прямоугольники (действия), ромбы (решения), параллельные дорожки (swimlanes).

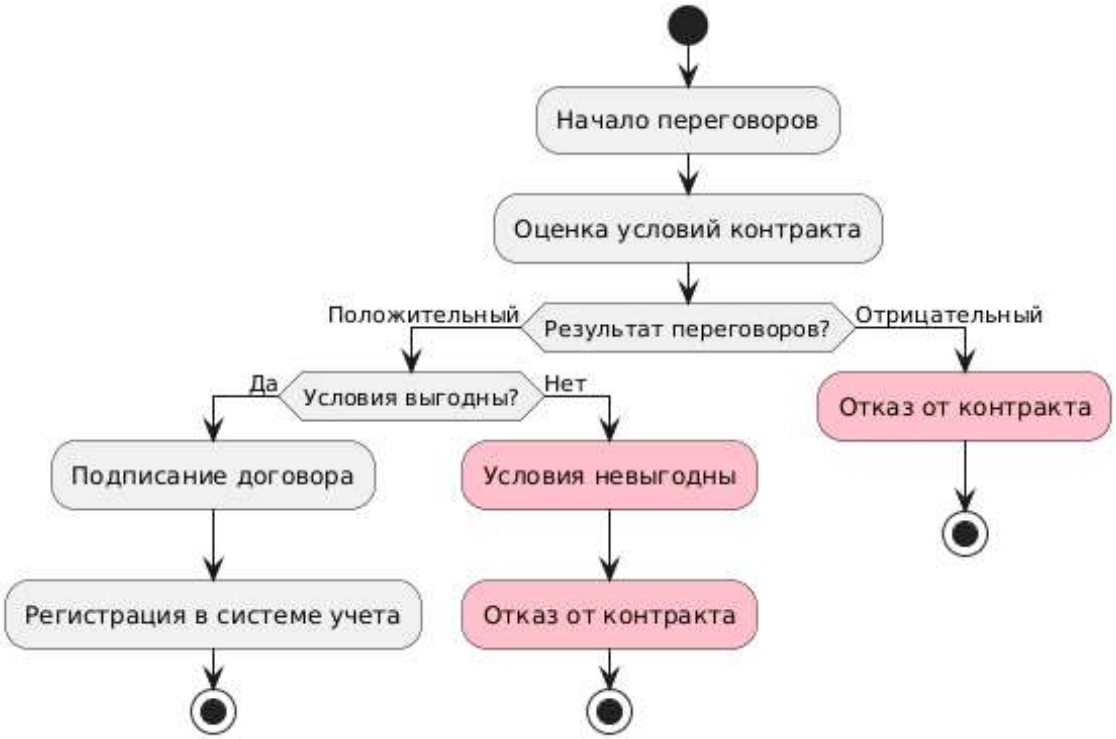
## Ошибки

- Не учитывать альтернативные сценарии (например, если проверка не пройдена).

## Как читать

- Следите за потоком действий и ответственностями (дорожки).

# Диаграмма активностей




```
@startuml
start
:Начало переговоров;
:Оценка условий контракта;
if (Результат переговоров?) then
(Положительный)
  if (Условия выгодны?) then (Да)
    :Подписание договора;
    :Регистрация в системе учёта;
    stop
  else (Нет)
    #Pink:Условия невыгодны;
   #Pink:Отказ от контракта;
  stop
  endif
else (Отрицательный)
  #Pink:Отказ от контракта;
  stop
endif
```

# Диаграмма состояний (поведенческая)

## Зачем

- Показать, как объект меняет состояния в ответ на события.

## Как выглядит

- Круги (начало/конец), прямоугольники с закруглёнными углами (состояния), стрелки (переходы).

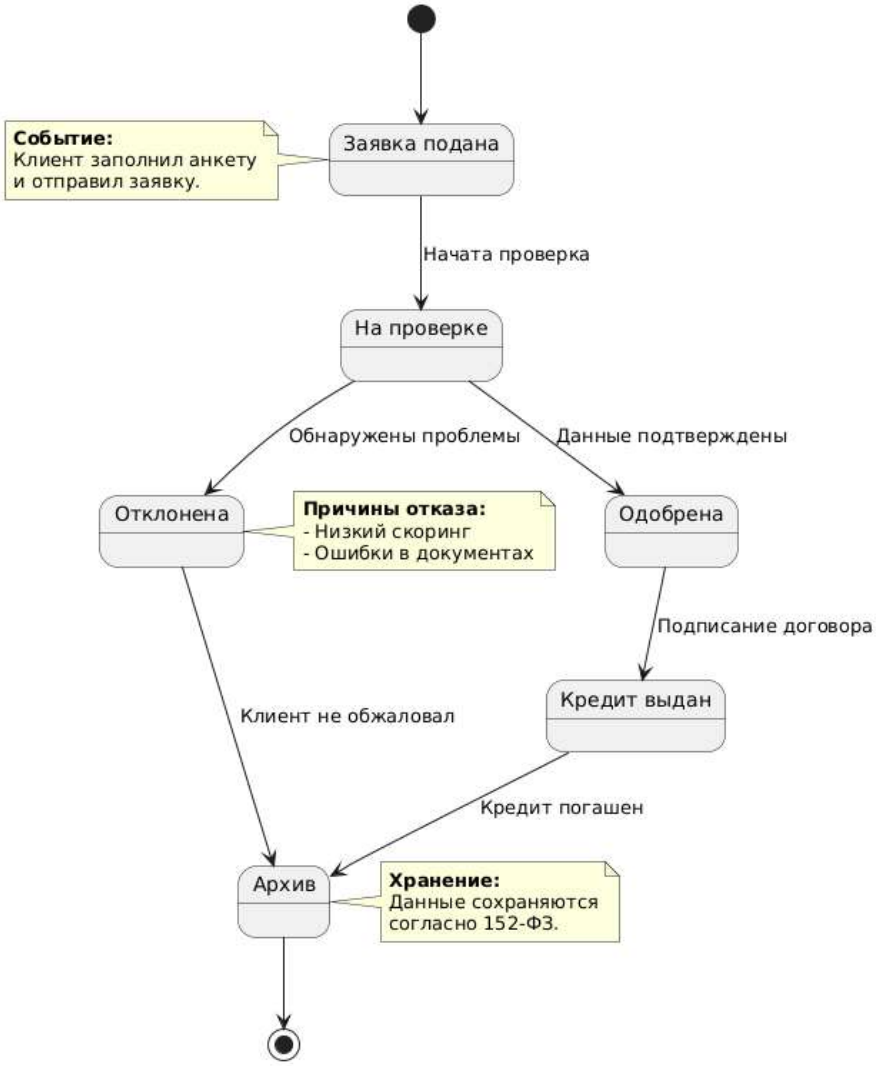
## Ошибки

- Не учитывать все возможные состояния (например, «Временная блокировка»).

## Как читать

- Отслеживайте возможные состояния объекта и триггеры переходов.

# Диаграмма состояний




```
@startuml
' Состояния кредитной заявки
state "Заявка подана" as submitted
state "На проверке" as checking
state "Одобрена" as approved
state "Отклонена" as rejected
state "Кредит выдан" as issued
state "Архив" as archived
' Начальное и конечные состояния
[*] --> submitted
submitted --> checking : Начата проверка
checking --> approved : Данные подтверждены
checking --> rejected : Обнаружены проблемы
approved --> issued : Подписание договора
issued --> archived : Кредит погашен
rejected --> archived : Клиент не обжаловал
archived --> [*]

' Заметки для пояснения
note left of submitted
  **Событие**
  Клиент заполнил анкету
  и отправил заявку
end note

note right of rejected
  **Причины отказа:**
  - низкий скоринг
  - ошибки в документах
end note
note right of archived
  **Хранение**
  Данные сохраняются
  согласно 152-ФЗ
end note
@enduml
```

# Диаграмма прецедентов — Use Case (поведенческая)

## Зачем

- Описать взаимодействие пользователей (акторов) с системой.

## Как выглядит

- Акторы (человечки) и Use Case / прецеденты (овалы) со связями.

## Ошибки

- Смешивать технические детали с бизнес-требованиями.

## Как читать

- Определите, какие действия доступны каждому актору.

# Диаграмма прецедентов (Use Case)


```
@startuml
left to right direction
actor Клиент
rectangle "Онлайн-банк" {
  (Перевести средства) as transfer
  (Открыть вклад) as deposit
  (Посмотреть баланс) as balance
  (Заблокировать карту) as block
}
Клиент --> transfer
Клиент --> deposit
Клиент --> balance
Клиент --> block
@enduml
```

# Диаграмма последовательностей (взаимодействия)

## Зачем

- Детализировать порядок сообщений между объектами во времени.

## Как выглядит

- Объекты (линии жизни), стрелки (сообщения), активационные полосы.

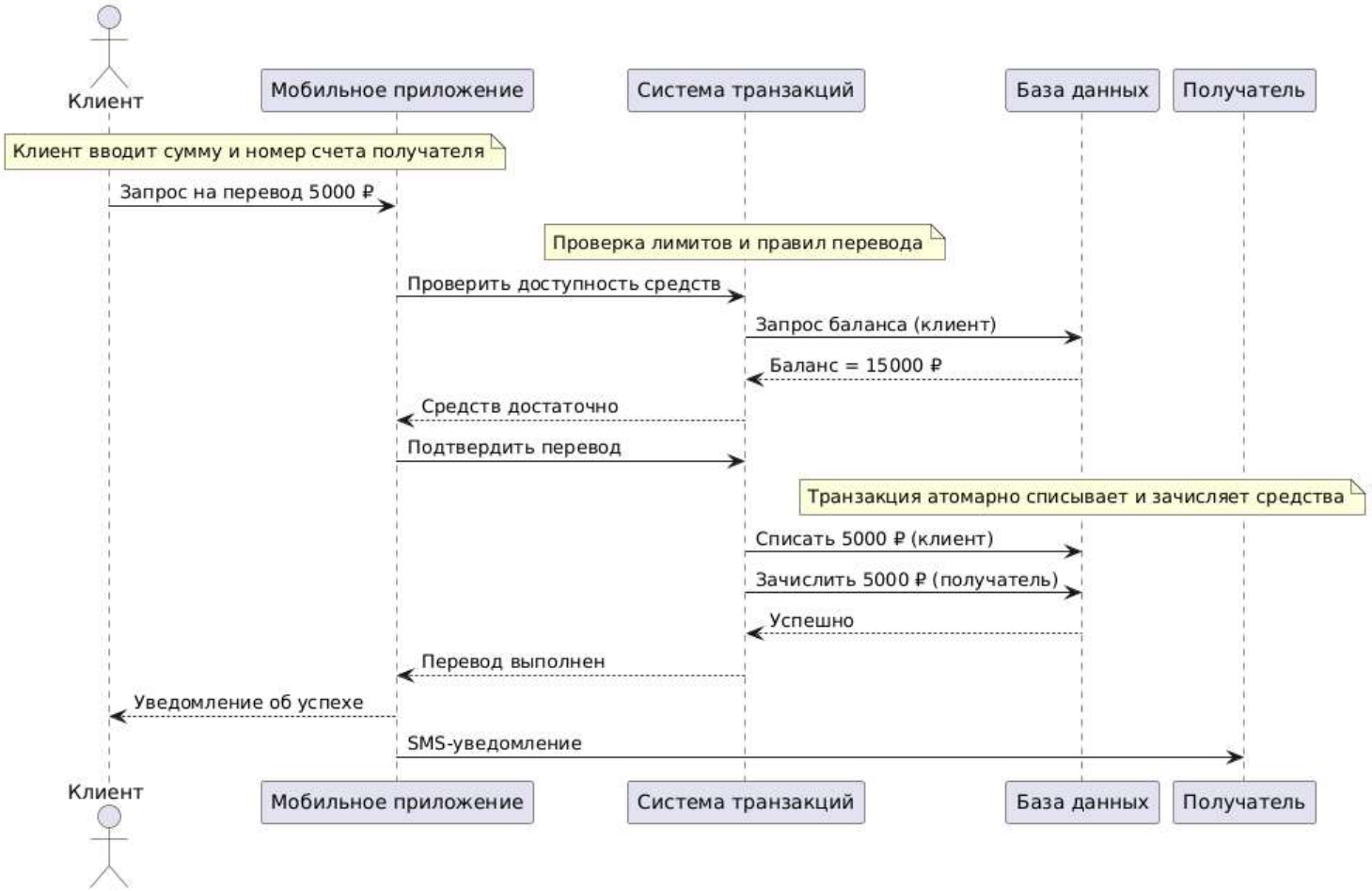
## Ошибки

- Не учитывать тайм-ауты или ошибки (например, «Сервер недоступен»).

## Как читать

- Следите за хронологией и участниками взаимодействия.

# Диаграмма последовательностей




```
@startuml

' Участники процесса
actor "Клиент" as client
participant "Мобильное приложение" as app
participant "Система транзакций" as auth
participant "База данных" as db
participant "Получатель" as recipient

' Процесс перевода
note over client, app: Клиент вводит сумму и номер счёта получателя
client -> app: Запрос на перевод 5 000 ₽
note over auth: Проверка лимитов и правил перевода
app -> auth: Проверить доступность средств
auth -> db: Запрос баланса (клиент)
db --> auth: Баланс = 15 000 ₽
auth --> app: Средств достаточно
app -> auth: Подтвердить перевод
note over db: Транзакция атомарно списывает и зачисляет средства
auth -> db: Списать 5 000 ₽ (клиент)
auth -> db: Зачислить 5 000 ₽ (получатель)
db --> auth: Успешно
auth --> app: Перевод выполнен
app --> client: Уведомление об успехе
app -> recipient: SMS-уведомление

@enduml
```

# Диаграмма коммуникации/общения (взаимодействия)

## Зачем

- Показать взаимодействие объектов в формате, акцентированном на связях.

## Как выглядит

- Объекты и сообщения между ними с нумерацией.

## Ошибки

- Путать с диаграммой последовательностей (здесь меньше акцента на время).

## Как читать

- Анализируйте роли объектов и последовательность сообщений.

# Диаграмма коммуникации (общения)


```
@startuml
left to right direction
skinparam linetype ortho
' Объекты системы
object "Клиент" as Client
object "Мобильное\nприложение" as App
object "Сервер" as Server
object "База данных" as DB

' Явное расположение объектов
Client -[hidden]-> App
App -[hidden]-> Server
Server -[hidden]-> DB

' Взаимодействие с нумерацией шагов
Client -> App : (1) Запрос баланса
App -> Server : (2) Проверить авторизацию
Server -> DB : (3) Получить баланс
DB --> Server : (4) Баланс = 5 000 ₽
Server --> App : (5) Данные клиента
App --> Client : (6) Показать баланс

‘ Заметки
note left of Client
Шаги:  1. Открывает раздел "Баланс"
6. Видит текущую сумму
end note
note right of DB
 Таблицы:
  - accounts
  - transactions
end note
@enduml
```

# Временная диаграмма (взаимодействия)

## Зачем

- Визуализировать изменение состояний объектов с учётом времени.

## Как выглядит

- Линии жизни с отметками времени и состояниями.

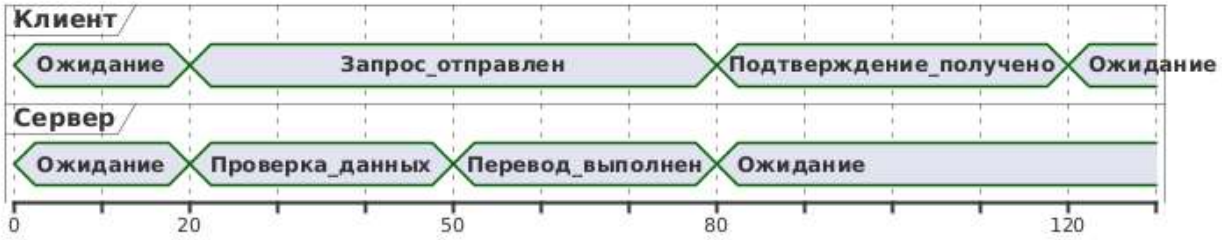
## Ошибки

- Не указывать критические временные рамки (например, максимальное время обработки).

## Как читать

- Обращайте внимание на временные ограничения и длительность состояний.

# Временная диаграмма




```
@startuml
concise "Клиент" as Client
concise "Сервер" as Server

' Начальное состояние
@0
Client is Ожидание
Server is Ожидание

' События и временные интервалы
@20
Client is Запрос_отправлен
Server is Проверка_данных

@50
Server is Перевод_выполнен

@80
Client is Подтверждение_получено
Server is Ожидание

@120
Client is Ожидание


@enduml
```

# Диаграмма обзора взаимодействий (взаимодействия)

## Зачем

- Комбинировать несколько диаграмм взаимодействия в одном представлении (например, последовательность + активность).

## Как выглядит

- Блок-схема со ссылками на другие диаграммы.

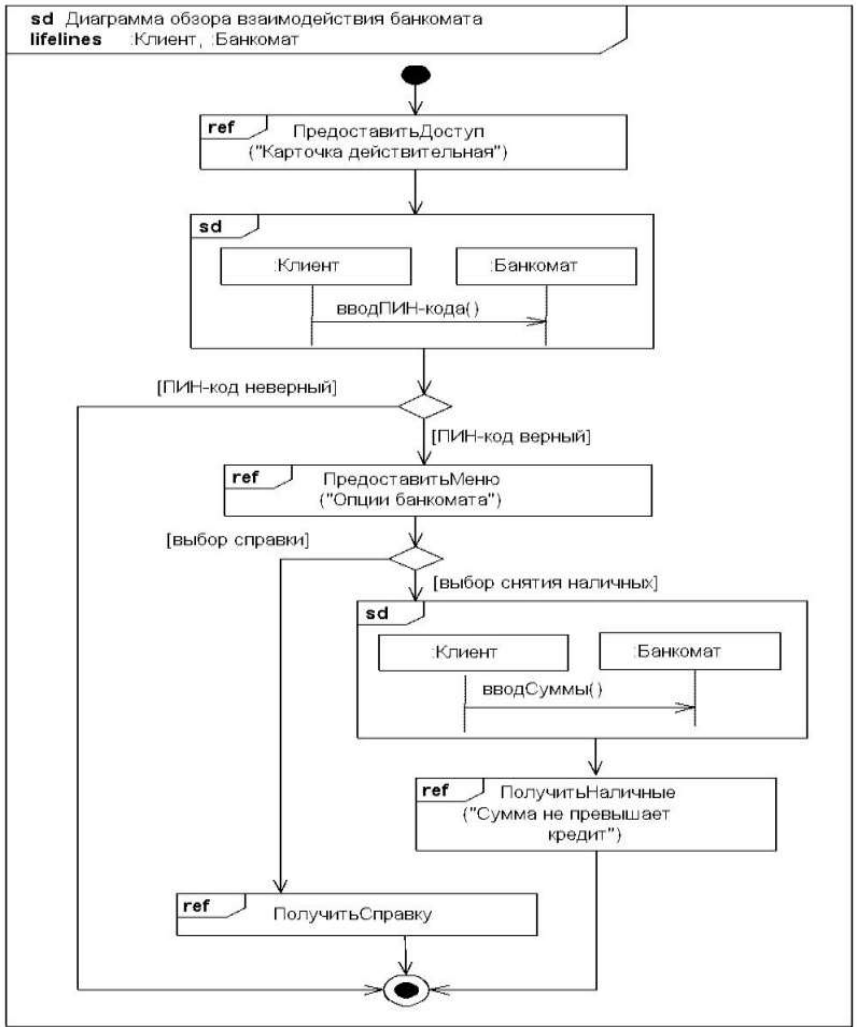
## Ошибки

- Перегружать диаграмму деталями вместо общего обзора.

## Как читать

- Используйте как карту для навигации между детализированными диаграммами.

# Диаграмма обзора взаимодействий




https://theslide.ru/uncategorized/UMLdiagrammy-diagramma-razvertyvaniya-i-prochie

# 3. Концептуальные области UML +I

# Концептуальная модель UML vs концептуальные области UML

**Концептуальная модель UML** — это базовый каркас языка UML, который определяет его ключевые элементы, правила их использования и связи между ними.

Это «метаописание» UML. Отвечает на вопросы:

- «Из каких строительных блоков состоит UML?»;
- «Как эти блоки сочетаются друг с другом?» (синтаксис и семантика).

**Концептуальные области UML** — это группировка инструментов UML по их назначению для решения конкретных задач проектирования.

Отвечают на вопрос «Какую часть системы мы описываем?».

# Концептуальная модель UML

Формальная основа языка, включающая:

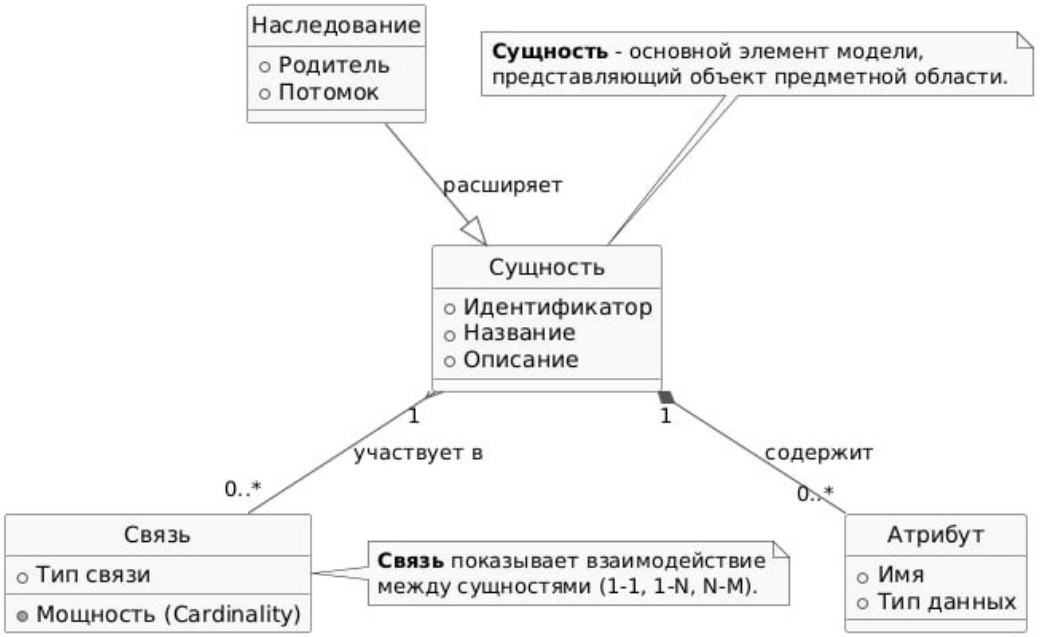
1. **Базовые элементы:**
   - сущности (Entities): классы, объекты, акторы, компоненты;
   - отношения (Relationships): ассоциации, наследование, зависимости;
   - поведенческие конструкции: сообщения, состояния, события.

**2. Абстрактные уровни модели:**

- метамодель UML — описание того, как определяются сами диаграммы;
- модель системы — конкретные диаграммы, созданные на основе метамодели.

**Пример:**

- метамодель определяет, что класс имеет атрибуты и методы;
- модель системы — класс «Счёт» с атрибутами «Номер» и «Баланс».



# Концептуальные области UML

Группы понятий и инструментов UML, которые помогают описывать разные аспекты системы:

+ статическую структуру;
+ динамическое поведение;
+ взаимодействия;
+ физическую реализацию;
+ архитектурные представления.

**Чтобы**:

+ упростить проектирование сложных систем;
+ унифицировать язык общения между командами.

# Область «Статическая структура»

## Описание компонентов системы и их связей

**Инструменты UML:**

+ диаграмма классов;
+ диаграмма объектов;
+ диаграмма компонентов;
+ диаграмма пакетов.

**Практическая польза:**

+ чётко определить, какие сущности существуют в системе (например, «Кредитная заявка», «Транзакция»);
+ увидеть зависимости между модулями.

**Пример:**

+ модель данных для клиентов и счетов (диаграмма классов);
+ группировка модулей системы в пакеты: «Клиенты», «Платежи», «Безопасность».

# Область «Динамическое поведение»

## Описание процессов и изменений в системе

**Инструменты UML:**

+ диаграмма активностей;
+ диаграмма состояний;
+ диаграмма прецедентов (Use Case).

**Практическая польза:**

+ оптимизировать бизнес-процессы (например, сократить время проверки клиента);
+ предотвратить ошибки в логике (например, неучтённые состояния карты).

**Пример:**

+ процесс «Одобрение кредита» (диаграмма активностей);
+ смена состояний счёта: «Активен» → «Заблокирован» → «Закрыт» (диаграмма состояний).

# Область «Взаимодействие объектов»

## Моделирование обмена сообщениями между компонентами

**Инструменты UML:**

+ диаграмма последовательностей;
+ диаграмма коммуникации;
+ временная диаграмма.

**Практическая польза:**

+ выявить узкие места в коммуникации (например, медленный ответ сервера);
+ убедиться, что процессы соответствуют SLA (например, обработка платежа за 2 секунды).

**Пример:**

+ сценарий «Оплата через мобильное приложение» (диаграмма последовательностей);
+ тайминги обработки транзакции (временная диаграмма).

# Область «Физическая реализация»

## Описание инфраструктуры и развёртывания системы

**Инструменты UML:**

+ диаграмма развёртывания;
+ диаграмма компонентов (частично).

**Практическая польза:**

+ понять, где хранятся данные (например, GDPR-требования для европейских клиентов);
+ спланировать масштабирование системы (например, добавление новых филиалов).

**Пример:**

+ архитектура системы: серверы баз данных, облачные хранилища, филиалы;
+ связи между мобильным приложением и API банка.

# Область «Архитектурные представления»

## Комплексное описание системы через разные ракурсы (по стандарту «4+1»)

**Представления:**

+ логическое (классы, объекты);
+ процессное (потоки, параллелизм);
+ физическое (развёртывание);
+ разработка (пакеты, модули);
+ сценарии использования (Use Case).

**Практическая польза:**

+ согласовать технические и бизнес-требования;
+ упростить документирование для аудита.

**Пример**

Полная архитектура мобильного банка:

+ логическое: классы «Пользователь», «Транзакция»;
+ физическое: серверы в AWS, шифрование данных;
+ сценарии: «Оплата QR-кодом», «Перевод между счетами».

# Связь концептуальных областей с диаграммами UML

| Область | Диаграммы | Пример применения |
|---|---|---|
| Статическая структура | Классов, объектов, компонентов | Модель данных для клиента и счёта |
| Динамическое поведение | Активностей, состояний, Use Case | Процесс одобрения ипотеки |
| Взаимодействие | Последовательностей, коммуникации | Авторизация платежа через 3D-Secure |
| Физическая реализация | Развёртывания | Схема серверов в ЦОД и филиалах |

# Как выбрать область?

**Сценарии использования**:

- если нужно описать данные → **статическая структура**;
- если важно смоделировать процесс → **динамическое поведение**;
- если требуется проанализировать интеграцию → **взаимодействие**;
- если обсуждаем инфраструктуру → **физическая реализация**.

**Пример**

Задача — оптимизировать процесс обработки заявок на кредит.

Используйте **динамическое поведение** (диаграммы активностей) + **взаимодействие** (последовательности).

# 4. Место UML в процессе разработки ПО

# Жизненный цикл разработки ПО и UML

# UML на этапе сбора требований

## Инструменты

**Диаграмма прецедентов** (Use Case) определяет, кто (акторы) и что (сценарии) делает в системе.
**Пример:** актор «Клиент» → Use Case «Открыть депозит», «Перевести деньги».

**Диаграмма активностей** детализирует бизнес-процессы.
**Пример:** процесс «Одобрение кредита» с ветвлениями (проверка кредитной истории, расчёт риска).

## Смысл:

+ чёткое понимание требований заказчика;
+ выявление противоречий на раннем этапе.

# UML на этапе проектирования

## Инструменты

**Диаграмма классов** моделирует структуру данных.
**Пример:** классы «Клиент», «Счёт», «Транзакция» с атрибутами и связями.

**Диаграмма последовательностей** показывает взаимодействие объектов.
**Пример:** оплата через мобильное приложение: Клиент → Приложение → Сервер → Банк-эквайер.

**Диаграмма состояний** описывает жизненный цикл объектов.
**Пример:** состояния карты «Активна» → «Заблокирована» → «Аннулирована».

## Смысл:

+ снижение риска архитектурных ошибок;
+ упрощение коммуникации в команде.

# UML на этапе реализации

## Инструменты

**Диаграмма компонентов** показывает модули системы и их интерфейсы.
**Пример:** компоненты «Платёжный шлюз», «API для СБП (Системы быстрых платежей)».

**Диаграмма развёртывания** описывает инфраструктуру.
**Пример:** серверы в ЦОД, облачные хранилища, подключение филиалов.

## Смысл:

+ контроль зависимостей между модулями;
+ планирование масштабируемости (например, добавление новых регионов).

# UML на этапах тестирования и сопровождения

## Инструменты

**Уточнение диаграмм:**

+ диаграммы последовательностей для тест-кейсов (например, «Ошибка при недостаточном балансе»);
+ диаграммы состояний для проверки всех сценариев блокировки карты.

**Документирование**

Актуализация диаграмм при изменении требований (например, добавление новых статусов транзакций).

## Смысл:

+ регрессионное тестирование на основе моделей;
+ быстрое внедрение изменений (например, обновление API для P2P-переводов).

# Гибкие методологии (Agile/Scrum) и UML

**Проблема**

UML считают избыточным для Agile.

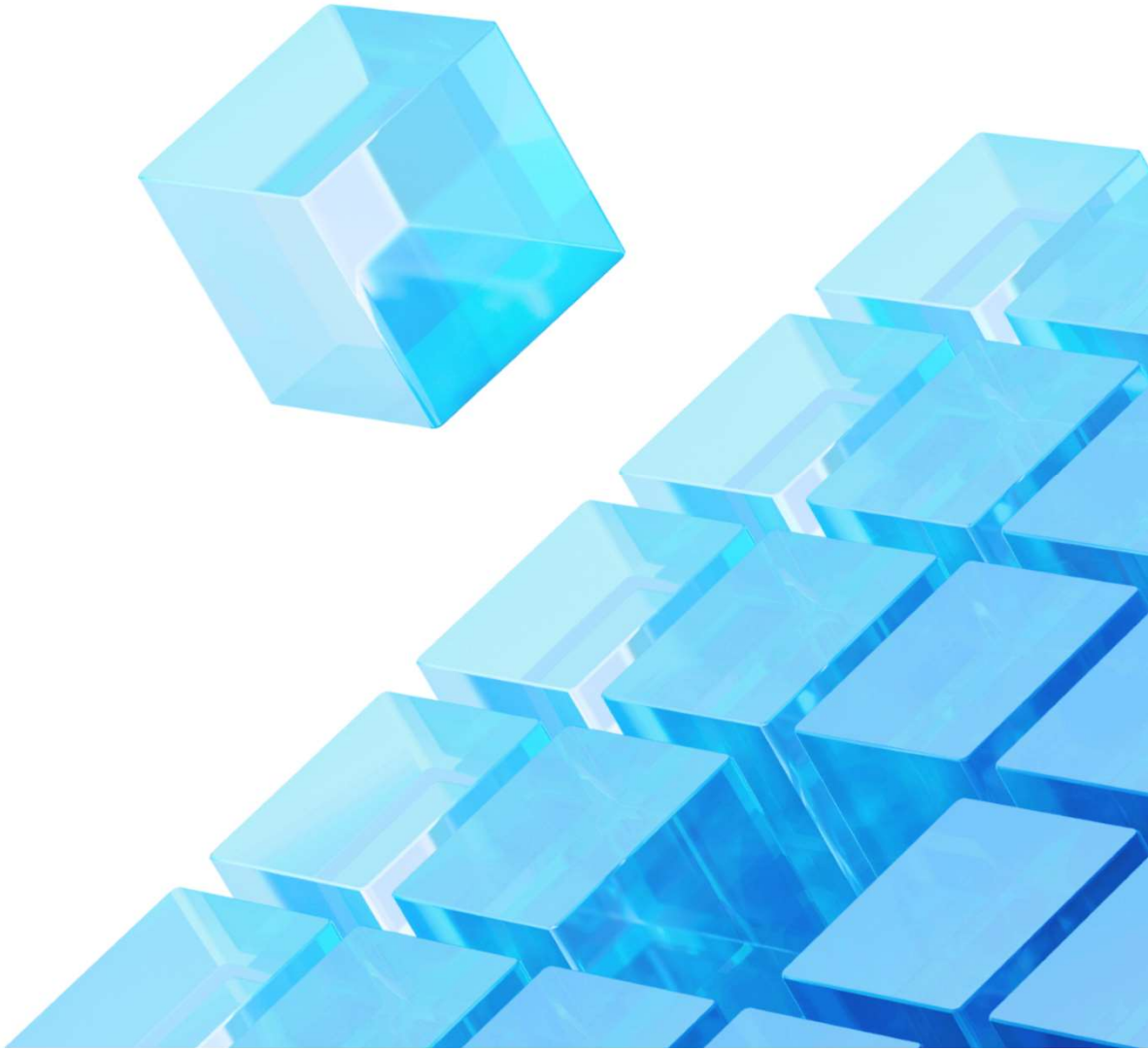
**Решение:**

+ использовать легковесные диаграммы (например, схематичные Use Case или последовательности);
+ фокус на ключевых артефактах:
  - спринт 1: Use Case для нового функционала (например, «Кредит под залог»);
  - спринт 2: диаграмма классов для данных о залоге.

**Смысл**

Баланс между гибкостью и документированием.

# Домашнее задание



# Домашнее задание

Выберите один из процессов в вашей текущей области деятельности.

**Определите:**

1. Какие 3–4 вида диаграмм UML лучше всего подойдут для описания процесса и почему (например, Use Case — для требований, последовательности — для взаимодействий).
2. К каким концептуальным областям UML относятся выбранные диаграммы (структура, поведение и т. д.).
3. На каких этапах ЖЦ ПО будет использоваться каждая диаграмма и как.

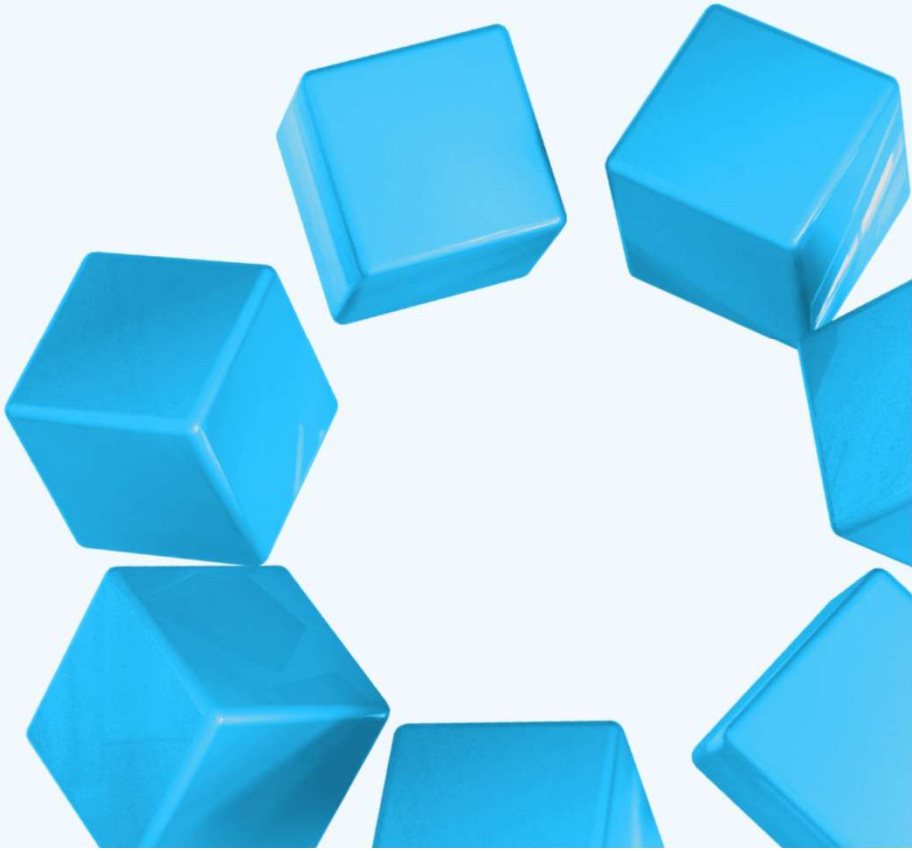
**Формат сдачи:**

- краткое описание процесса (для понимания проверяющим преподавателем);
- диаграммы;
- области;
- использование.

# Спасибо
# за внимание

@COACH GOLOVANOVA

telegram:
@valentina_pro_coach



TI