

UML диаграммы классов на примере PlantUML и Mermaid

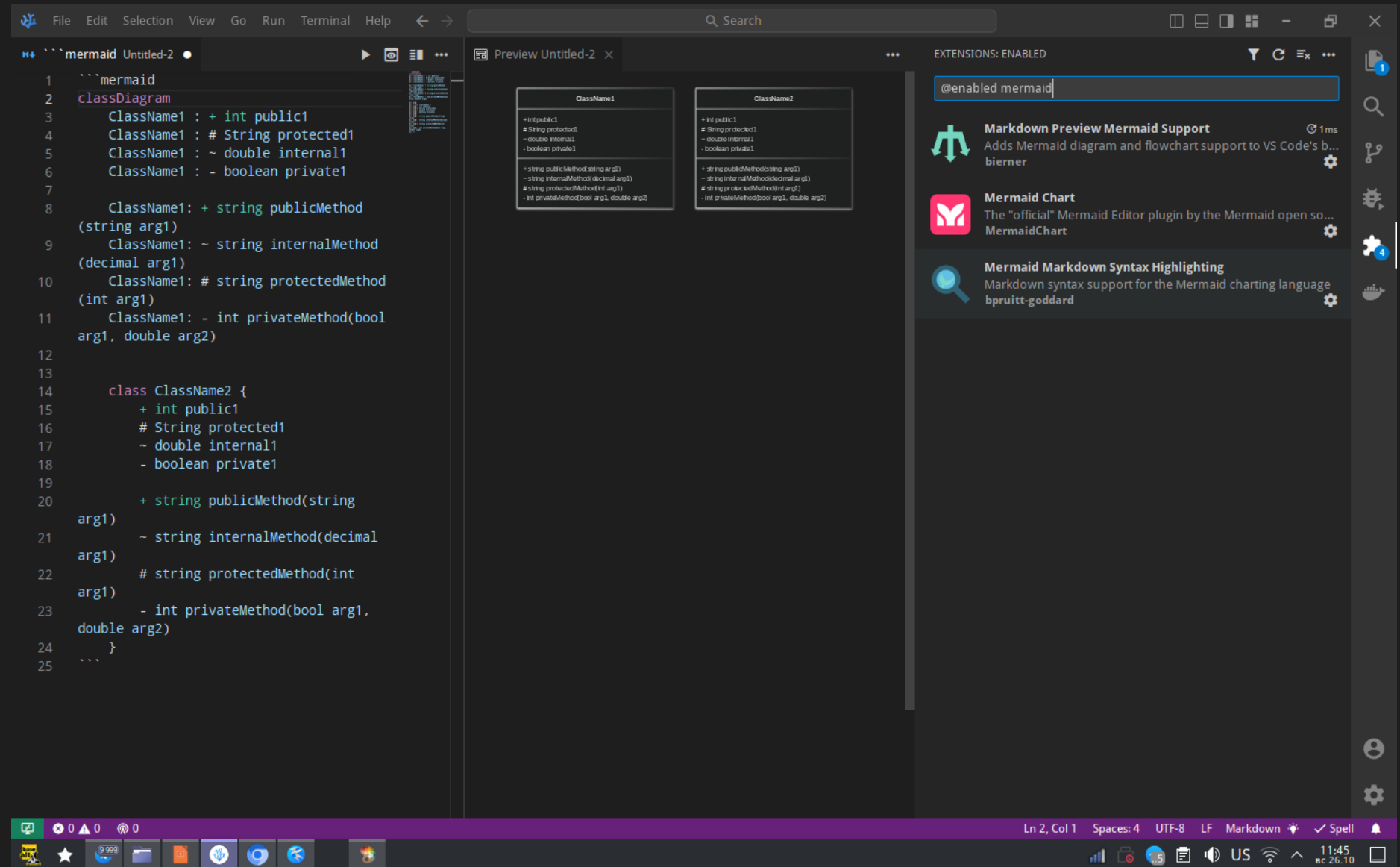
Автор: Солопий А. В.

Аспирант КНИТУ-КАИ

Visual Studio Code. Плагин PlantUML

Требуется установка:

Markdown Preview Mermaid
Mermaid Chart
Mermaid Syntax Highlighting



Visual Studio Code. Mermaid. Плагин Markdown

Требуется установка в системе:

Java
GraphViz

The screenshot shows the Visual Studio Code interface. The main editor displays the PlantUML extension page, which includes the extension's logo, name (PlantUML v2.18.1), author (jebbs), download count (96,710), and a 5-star rating. The extension is described as "Rich PlantUML support for Visual Studio Code." and is currently enabled globally. The page also features a "PlantUML README" section with a notice about the server render and a list of features. The right sidebar shows the "EXTENSIONS: ENABLED" list, with the PlantUML extension listed as "@enabled uml". The bottom status bar shows the system tray with various icons and the time (11:35).

File Edit Selection View Go Run Terminal Help Search

@startuml Диаграмма состояний Untitled-1 Extension: PlantUML

EXTENSIONS: ENABLED

@enabled uml

PlantUML v2.18.1
jebbs | 96,710 | ★★★★★(1)
Rich PlantUML support for Visual Studio Code.
Disable Uninstall ⌵ ⚙️
This extension is enabled globally.

DETAILS FEATURE CONTRIBUTIONS CHANGELOG RUNTIME STATUS

PlantUML README

Rich PlantUML support for Visual Studio Code.

build unknown

Notice

The server render is now very recommended by this extension, since it's much faster and much easier to setup, while the major weakness has been improved:

- Enables **POST** method, you can now render very-large diagrams
- The extension comes with a new **include processor**, you won't encounter include problems again.

To make sure you can benefit from this update, make sure your plantuml server enables **POST** support.

If not, like the official <https://www.plantuml.com/plantuml>, the extension falls back to use **GET**, and you may still encounter **414 URI Too Long** errors. Setup your own server is recommended.

See also: [About Render](#)

Features

Categories

Programming Languages
Formatters Snippets
Other

Extension Resources

[Marketplace](#)
[Repository](#)
[License](#)
[jebbs](#)

More Info

Published 2020-8-7, 01:49:19
Last released 2024-4-12, 06:30:12
Last updated 2025-5-9, 20:55:15
Identifier jebbs.plantuml

Spell

11:35 BC 26.10

Диаграммы классов

Диаграмма классов. Объявление.

PlantUML

@startuml

'определение класса

class ClassName

{

+ int public2

'публичное свойство (поле, атрибут)

+ public1: int

'защищённое свойство (видимость наследникам)

protected1: string

'внутреннее свойство (видимость уровня модуля)

~ internal1: double

'приватное свойство (видимость уровня модуля)

- private1: boolean

'публичный метод

+ publicMethod(arg1: string) : string

'защищённый метод

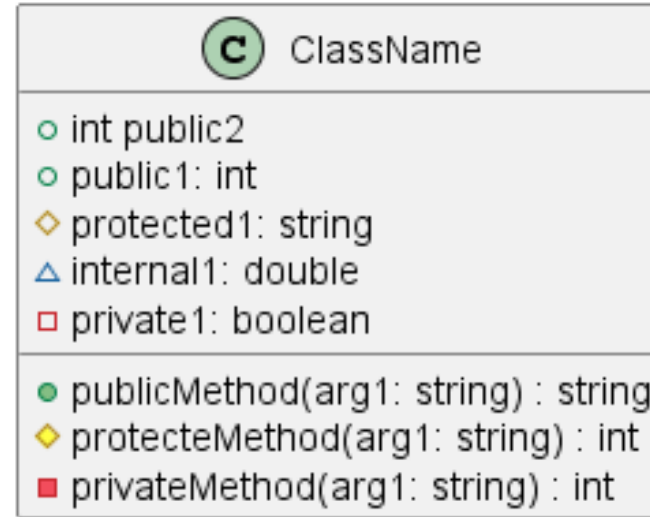
protecteMethod(arg1: string) : int

'приватный метод

- privateMethod(arg1: string) : int

}

@enduml



''' определение класса

''' class <Name> {}

'''

''' определение поля (1 способ)

''' [+|#|~|-] <Name> : <Type>

'''

''' определение поля (2 способ)

''' [+|#|~|-] <Type> <Name>

Диаграмма классов. Объявление.

@startuml

'определение класса

'публичное свойство (поле, атрибут)

ClassName : + int public2

'защищённое свойство (видимость наследникам)

ClassName : # protected1: string

'внутреннее свойство (видимость уровня модуля)

ClassName : ~ internal1: double

'приватное свойство (видимость уровня модуля)

ClassName : - private1: boolean

'публичный метод

ClassName : + publicMethod(arg1: string) : string

'защищённый метод

ClassName : # protecteMethod(arg1: string) : int

'внутренний метод

ClassName : - internalMethod(arg1: string) : int

'приватный метод

ClassName : - privateMethod(arg1: string) : int

''' определение класса

''' class <Name> {}

'''

''' определение поля (1 способ)

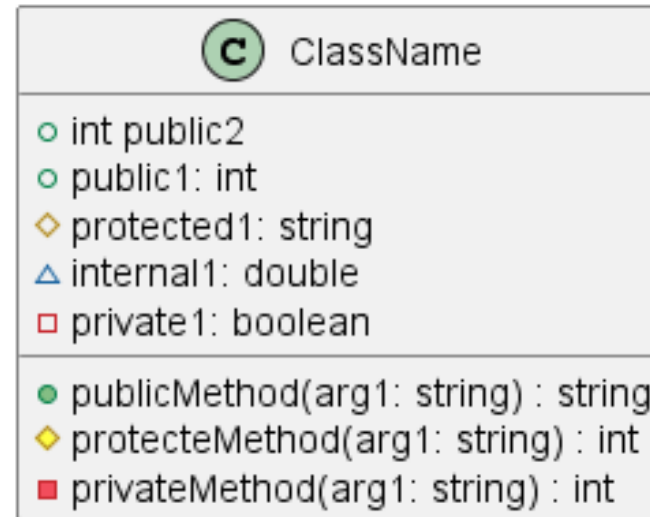
''' [+|#|~|-] <Name> : <Type>

'''

''' определение поля (2 способ)

''' [+|#|~|-] <Type> <Name>

@enduml



''' определение класса

''' class <Name> {}

'''

''' определение поля (1 способ)

''' [+|#|~|-] <Name> : <Type>

'''

''' определение поля (2 способ)

''' [+|#|~|-] <Type> <Name>

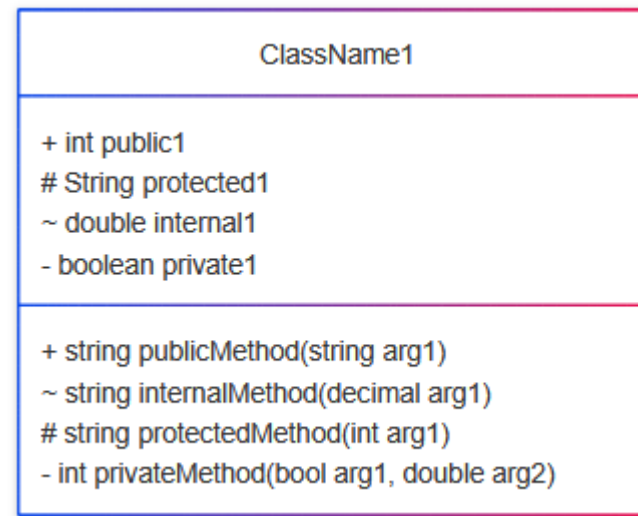
Диаграмма классов. Объявление. Mermaid

```
---
config:
  theme: mc
---
classDiagram
  ClassName1 : + int public1
  ClassName1 : # String protected1
  ClassName1 : ~ double internal1
  ClassName1 : - boolean private1

  ClassName1: + string publicMethod(string arg1)
  ClassName1: ~ string internalMethod(decimal arg1)
  ClassName1: # string protectedMethod(int arg1)
  ClassName1: - int privateMethod(bool arg1, double
arg2)
```

```
class ClassName2 {
  + int public1
  # String protected1
  ~ double internal1
  - boolean private1

  + string publicMethod(string arg1)
  ~ string internalMethod(decimal arg1)
  # string protectedMethod(int arg1)
  - int privateMethod(bool arg1, double arg2)
}
```



```
%% определение класса
%% class <Name> {}
%%
%% определение класса
%% <ClassName> : [+|#|~|-] <Type> <Name>
%%
%% определение поля
%% [+|#|~|-] <Type> <Name>
%%
%% определение метода
%% [+|#|~|-] <Type> <Name>()
```

Диаграмма классов. Уровни ВИДИМОСТИ

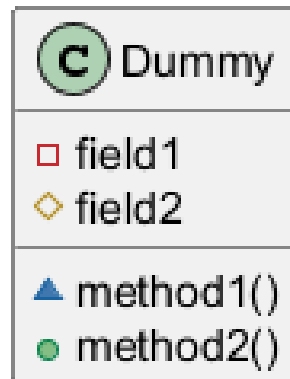
Plant UML

Символ	Иконка для поля данных	Иконка для метода	Видимость
-	□	■	private
#	◇	◆	protected
~	△	▲	package private
+	○	●	public

@startuml

```
class Dummy {
  -field1
  #field2
  ~method1()
  +method2()
}
```

@enduml



Mermaid

To describe the visibility (or encapsulation) of an attribute or method/function that is a part of a class (i.e. a class member), optional notation may be placed before that members' name:

- + Public
- Private
- # Protected
- ~ Package/Internal

```
---
config:
  theme: mc
---
classDiagram
class Dummy {
  - field1
  # field2
  ~ method1()
  + method2()
}
```



Диаграмма классов. Абстрактные и статические классы. PlantUML

@startuml

left to right direction

'определение класса

class ClassName

{

'публичное статическое свойство (поле, атрибут)

{static} + propStatic: int

'публичный статический метод

{static} + methodStatic(arg1: int, arg2: string): void

}

'определение класса

abstract class AbstractClassName

{

'публичное абстрактное свойство (поле, атрибут)

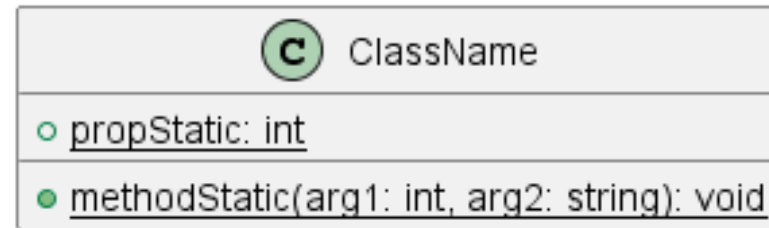
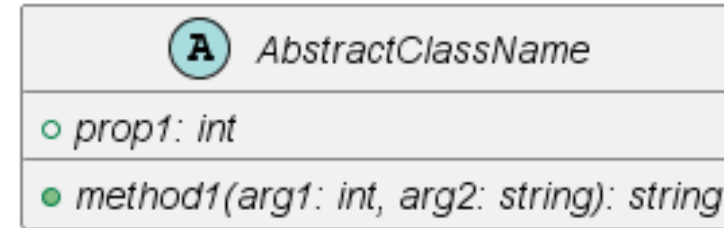
{abstract} + prop1: int

'публичный абстрактный метод

{abstract} + method1(arg1: int, arg2: string): string

}

@enduml



''' определение абстрактного класса

''' abstract class <Name> {}

'''

''' определение статического поля

''' {static} [+|#|~|-] <Name> : <Type>

'''

''' определение абстрактного поля

''' {abstract} [+|#|~|-] <Name> : <Type>

Диаграмма классов. Абстрактные и статические классы. Mermaid

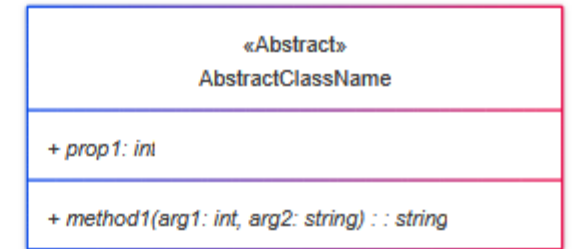
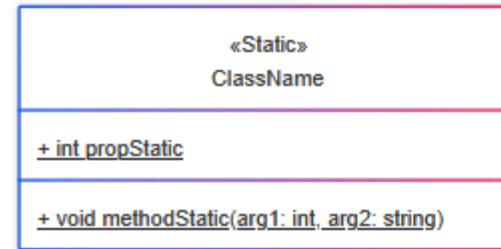
```
---
config:
  theme: mc
---
classDiagram

%% определение класса и статические элементы
class ClassName {
  %% публичное статическое свойство (поле, атрибут)
  + int propStatic$

  %% публичный статический метод
  + void methodStatic(arg1: int, arg2: string)$
}

%% определение абстрактного класса
class AbstractClassName {
  <<Abstract>>
  %% публичное абстрактное свойство (поле, атрибут)
  + prop1: int*

  %% публичный абстрактный метод
  + method1(arg1: int, arg2: string): string*
}
```



note you can also include additional *classifiers* to a method definition by adding the following notation to the *end* of the method, i.e.: after the () or after the return type:

- * Abstract e.g.:

- someAbstractMethod()* or

- someAbstractMethod() int*

- \$ Static e.g.: someStaticMethod()\$ or

- someStaticMethod() String\$

note you can also include additional *classifiers* to a field definition by adding the following notation to the very end:

- \$ Static e.g.: String someField\$

Диаграмма классов. Интерфейсы и перечисления. PlantUML

@startuml

left to right direction

'определение перечисления

enum EnumName

```
{
    One = 1
    Two = 2
    Three = 3
}
```

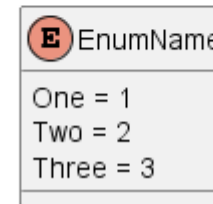
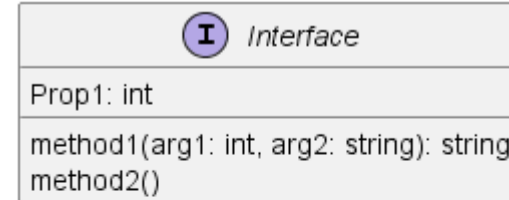
'определение интерфейса

interface Interface

```
{
    'свойство (поле, атрибут)
    Prop1: int

    'публичный абстрактный метод
    method1(arg1: int, arg2: string): string
    method2()
}
```

@enduml



''' определение перечисления

''' enums <Name> {}

'''

''' определение интерфейса

''' interface <Name> {}

'''

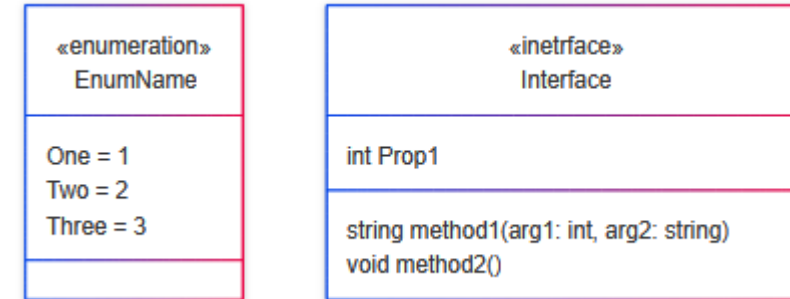
Диаграмма классов. Интерфейсы и перечисления. Mermaid

```
---
config:
  theme: mc
---
classDiagram

class EnumName {
  <<enumeration>>
  One = 1
  Two = 2
  Three = 3
}

class Interface {
  <<interface>>
  %%свойство (поле, атрибут)
  int Prop1

  %%публичный абстрактный метод
  string method1(arg1: int, arg2: string)
  void method2()
}
```



```
%% определение перечисления
%% class <Name> {
%%     <<enumeration>>
%% }
%%
%% определение интерфейса
%% class <Name> {
%%     <<interface>>
%% }
%%
```

Диаграмма классов. Наследование и реализация интерфейса. PlantUML

```
@startuml
    <<constructor>>
        + GetName() : string
        + SetWorld(world: World)
        + Render()
    }
    class SuperSpecificClass
    {
        - details : map<int, string>
        + Print(fast: bool)
    }
    TypeBase <|-- SpecificClass
    IRenderable <|.. SpecificClass
    World <-- SpecificClass

    SpecificClass <|--
    SuperSpecificClass
    IPrintable <|.. SuperSpecificClass

@enduml

class World
{
}

class SpecificClass
{
    - _world: World
    # options: string[]
}
```

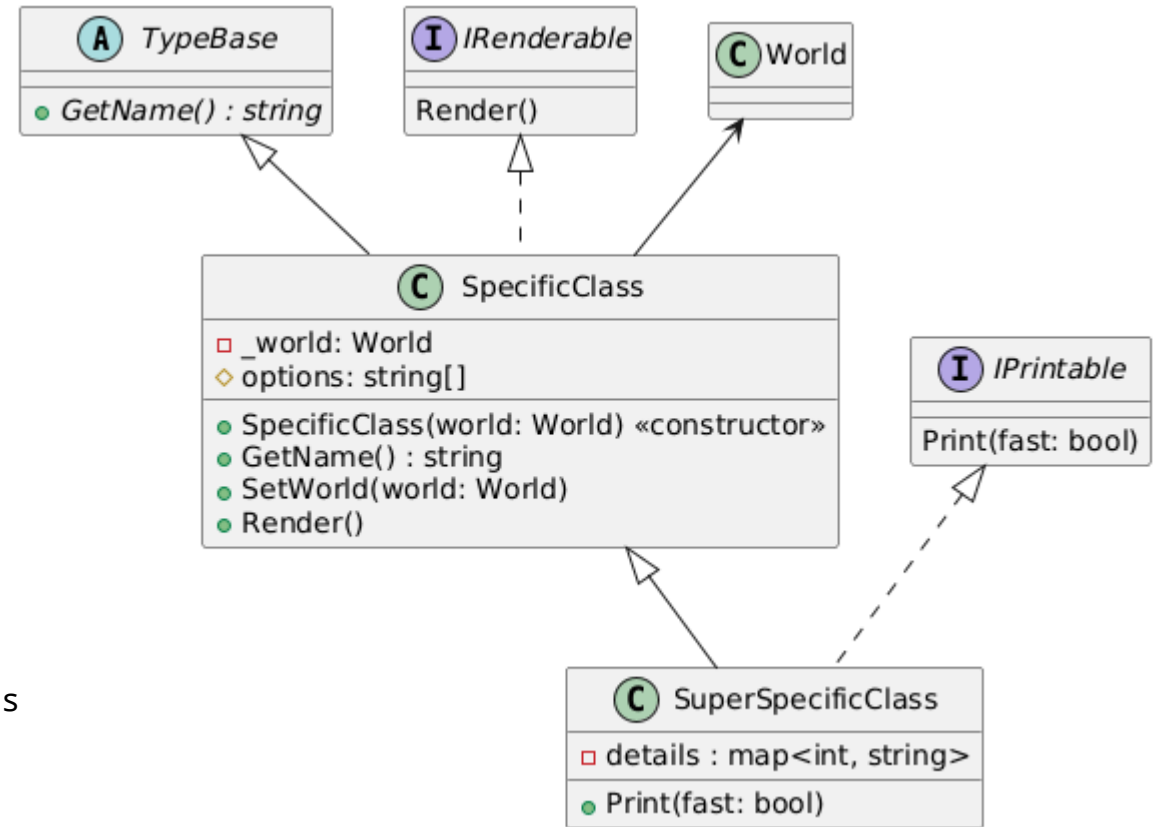


Диаграмма классов. Наследование и реализация интерфейса. Mermaid

```
---
config:
  theme: mc
---
classDiagram
    class TypeBase {
        <<Abstract>>
        + string GetName()
    }
    class IRenderable {
        <<Interface>>
        Render()
    }
    class IPrintable {
        <<Interface>>
        Print(boolfast)
    }
    class World
    class SpecificClass {
        - _world World
        # string[] options
        + SpecificClass(World world)
        + string GetName()
        + SetWorld(World world)
        + Render()
    }
    class SuperSpecificClass {
        - map details
        + Print(bool fast)
    }
    TypeBase <|-- SpecificClass
    IRenderable <|.. SpecificClass
    World <|-- SpecificClass
    IPrintable <|.. SuperSpecificClass
    SuperSpecificClass <|.. SpecificClass
```

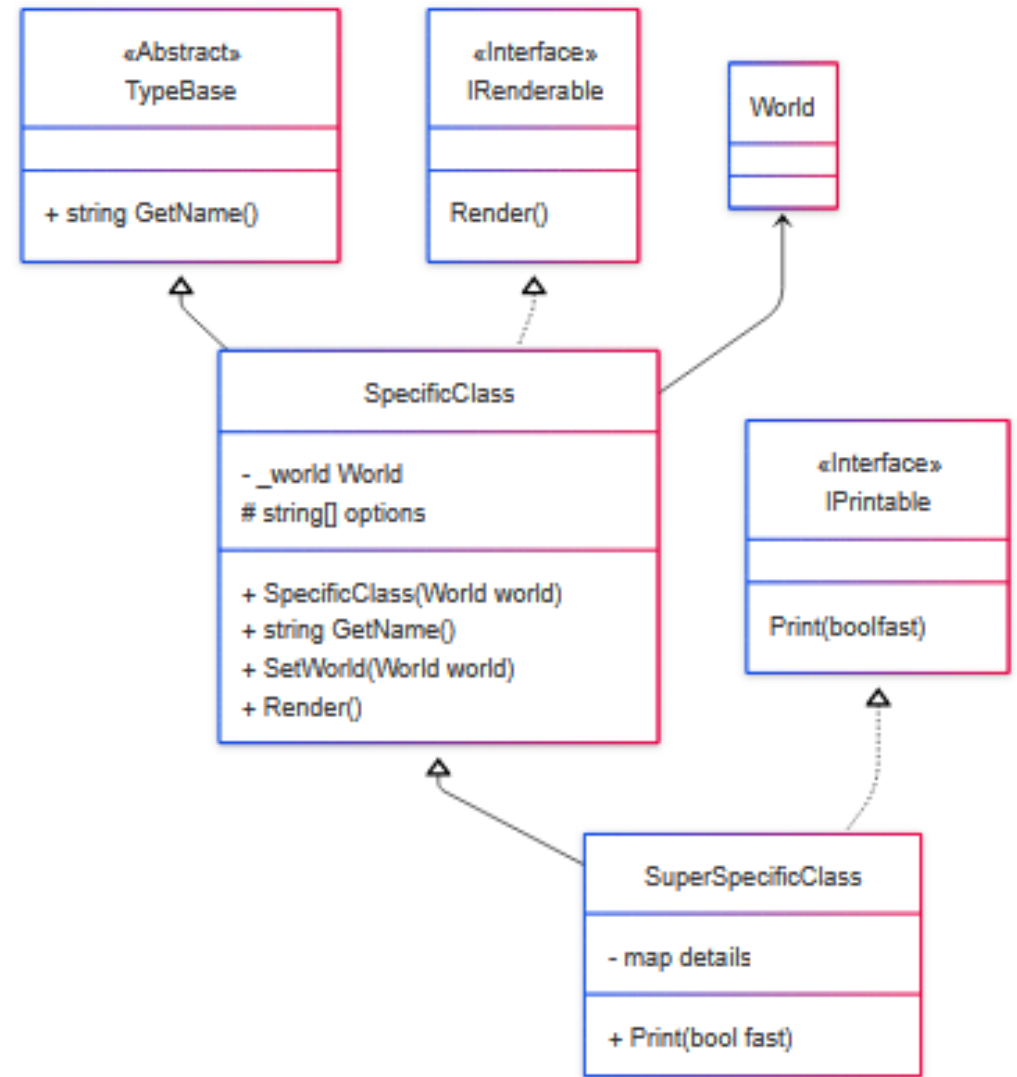


Диаграмма классов. Связи. PlantUML

```
@startuml
class A
class B

class AX
class BX {
    # b: string
    + a: int
}

class A1
class B1 {
    + a1: int
    # b1: string
}

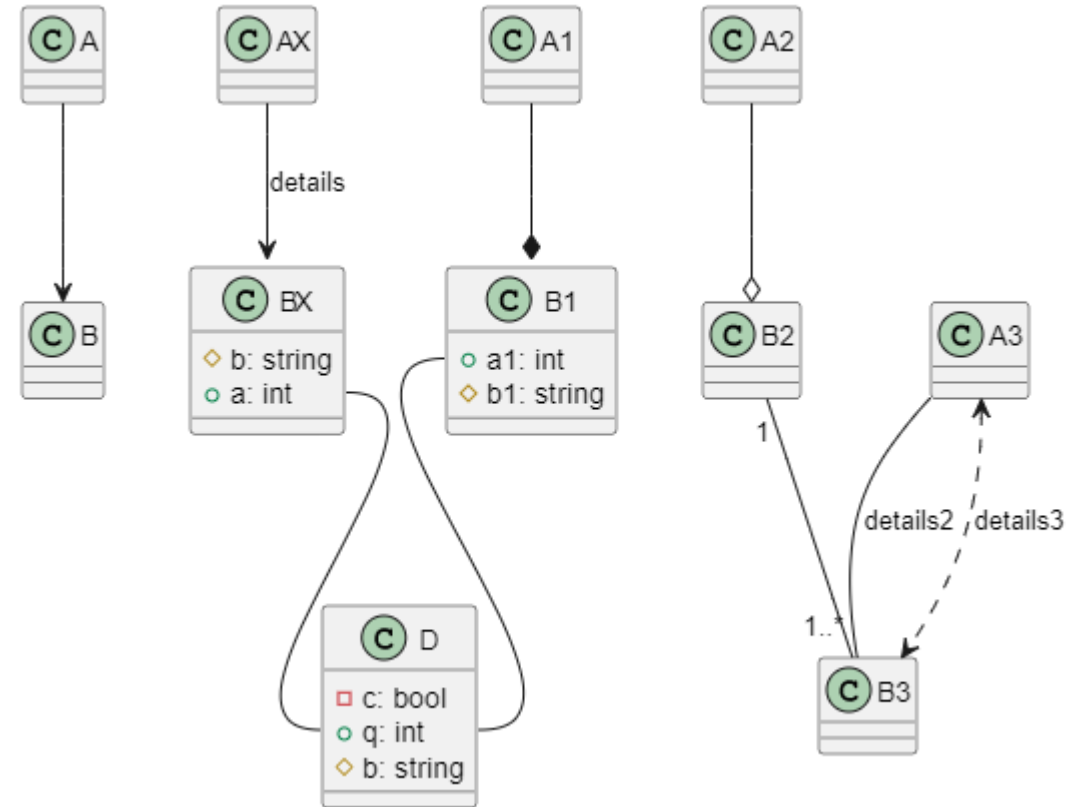
class A2
class B2

class A3
class B3

class D {
    - c: bool
    + q: int
    # b: string
}
```

```
' КОМПОЗИЦИЯ
A1 --* B1
' агрегация
A2 --o B2
' СВЯЗЬ
A3 -- B3 : details2
' СВЯЗЬ СО СТРЕЛКАМИ
A3 <..> B3 : details3
' СВЯЗЬ С РАЗМЕРНОСТЯМИ
B2 "1" -- "1..*" B3

' СВЯЗЬ С ПОЛЯМИ
BX::a -- D::q
B1::a1 -- D::q
@enduml
```



```
' ассоциация
A --> B
' ассоциация с деталями
```

Диаграмма классов. Связи. Mermaid

```
classDiagram
```

```
class A
class B
```

```
class AX
class BX {
    # b: string
    + a: int
}
```

```
class A1
class B1 {
    + a1: int
    # b1: string
}
```

```
class A2
class B2
```

```
class A3
class B3
```

```
class D {
    - c: bool
    + q: int
    # b: string
}
```

```
%% ассоциация
```

```
A --> B
```

```
%% ассоциация с деталями
```

```
AX --> BX : details
```

```
%% композиция
```

```
A1 --* B1
```

```
%% агрегация
```

```
A2 --o B2
```

```
%% связь
```

```
A3 -- B3 : details2
```

```
%% связь со стрелками
```

```
A3 <..> B3 : details3
```

```
%% связь с размерностями
```

```
B2 "1" -- "1..*" B3
```

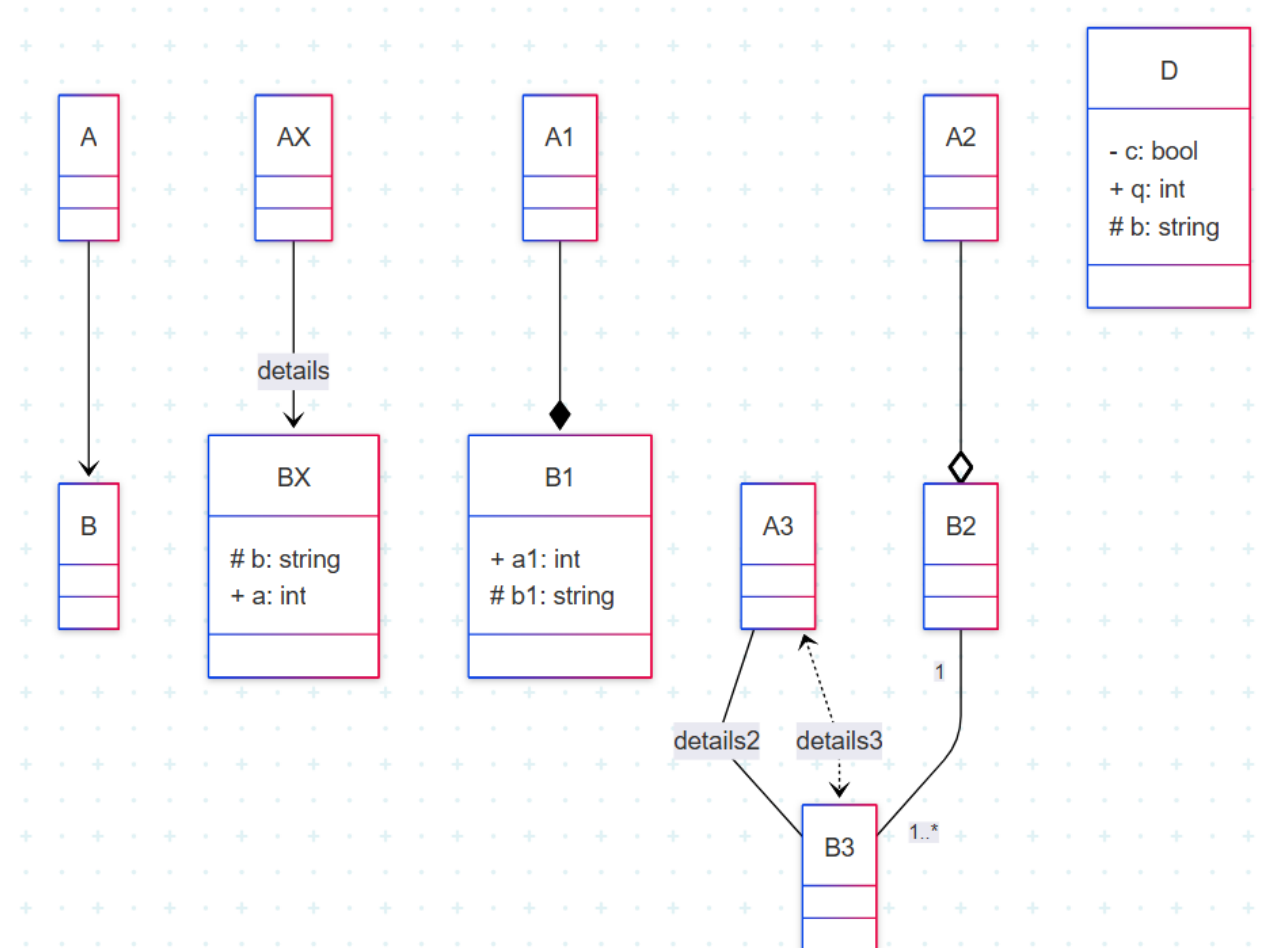


Диаграмма классов. Типы связей

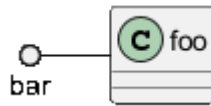
Plant UML

Тип	Символ	
Расширение	< --	К любой стороне (или сразу к обоим сторонам) линии, соединяющей два элемента, можно добавить разные наконечники:
Реализация	< ..	<ul style="list-style-type: none"> < - рисует заостренный наконечник стрелочки < или ^ - рисует наконечник стрелочки в виде треугольника * - рисует наконечник стрелочки в виде сплошного ромба o - рисует наконечник стрелочки в виде полого ромба # - рисует наконечник стрелочки в виде полого квадрата x - рисует наконечник стрелочки в виде крестика }- рисует наконечник стрелочки в виде обратного треугольника + - рисует наконечник стрелочки в виде кружочка с крестиком внутри
Композиция	*--	
Агрегация	o--	
Зависимость	-->	
Зависимость	..>	

есть два типа линий:

- - рисует сплошную линию
- .. - рисует штриховую линию

```
@startuml
class foo
bar ()-
foo
@enduml
```



Mermaid

Type	Description
<	Наследование
*	Композиция
o	Агрегация
>	Ассоциация
<	Ассоциация
>	Реализация

Тип линии	Описание
--	Сплошная
..	Точки

```
classDiagram
classA --|> classB : Inheritance
classC --* classD : Composition
classE --o classF : Aggregation
classG --> classH : Association
classI -- classJ : Link(Solid)
classK ..> classL : Dependency
classM ..|> classN : Realization
classO .. classP : Link(Dashed)
```

```
classDiagram
bar ()--
foo
```

bar

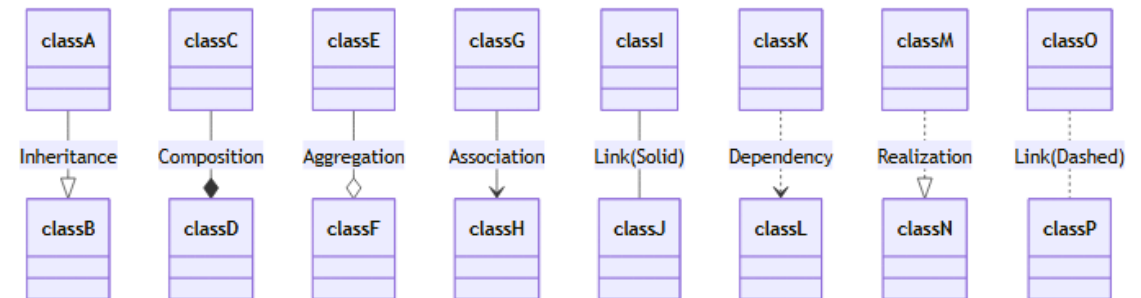


Диаграмма классов. Дженерики. PlantUML

@startuml

```
interface IList<T extends class> {  
    + Count: int  
    + Add(item: T)  
}
```

```
class List<T extends class> {  
    + Count: int  
    + Add(item: T)  
}
```

```
class StringList {  
  
}
```

```
IList <|.. List  
List <|-- StringList
```

@enduml

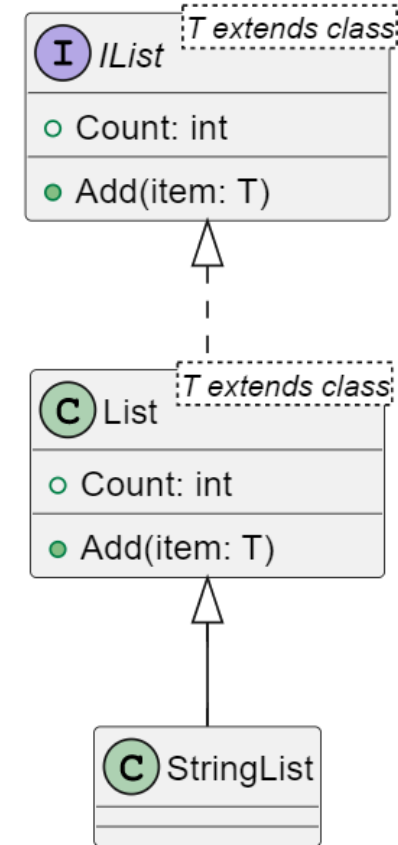


Диаграмма классов. Дженерики. Mermaid

```
classDiagram
direction TB
class IList~T~ {
+ int Count
+ Add(T item)
}
class List~T~ {
+ int Count
+ Add(T item)
}
class StringList {
+ GenericMethod~G~(G arg)
}

IList <|-- List
List <|-- StringList
```

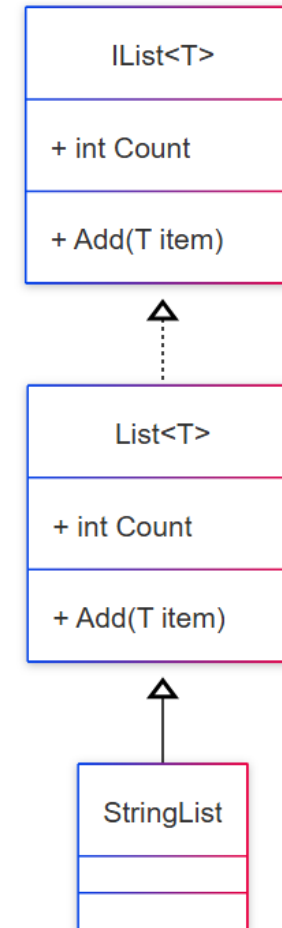
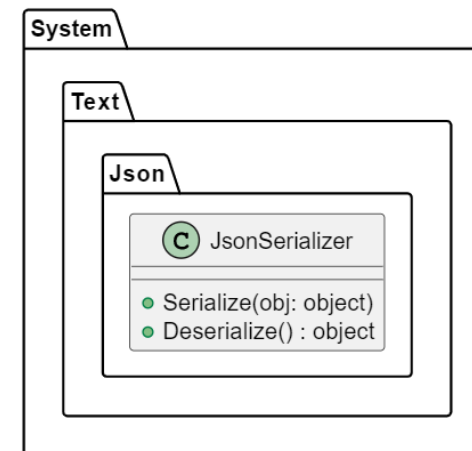


Диаграмма классов. Пространства имён. PlantUML

@startuml

```
set namespaceSeparator ::
class System::Text::Json::JsonSerializer {
    + Serialize(obj: object)
    + Deserialize() : object
}
```

@enduml



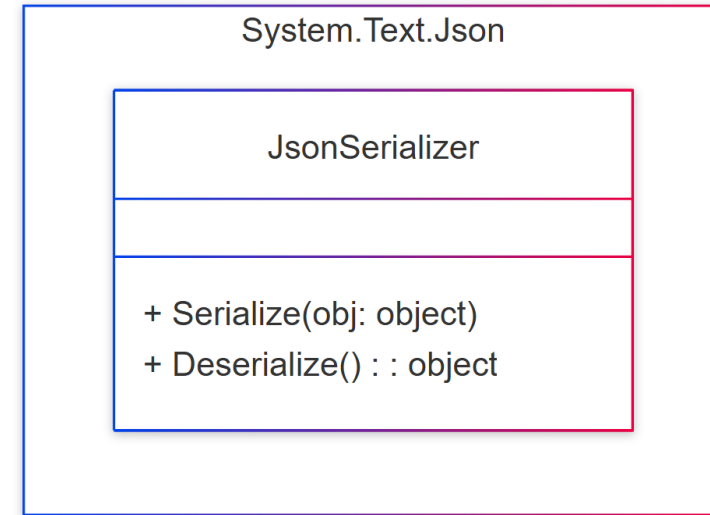
@startuml

```
namespace System {
    namespace Text {
        namespace Json {
            class
System::Text::Json::JsonSerializer {
    + Serialize(obj: object)
    + Deserialize() : object
        }
    }
}
}
```

@enduml

Диаграмма классов. Пространства имён. Mermaid

```
classDiagram
namespace System.Text.Json {
  class JsonSerializer {
    + Serialize(obj: object)
    + Deserialize() : object
  }
}
```



Диаграммы состояний

Диаграмма состояний. PlantUML

@startuml

```
[*] --> Ожидание
Ожидание --> [*]
Ожидание --> Запуск
Запуск --> Разогрев
Разогрев --> Ожидание : недостаток
ресурса
Разогрев --> Полёт : ресурс
имеется
Полёт --> [*]
```

@enduml



Диаграмма состояний. Mermaid

Диаграмма состояний

stateDiagram-v2

```
[*] --> Ожидание
Ожидание --> [*]
Ожидание --> Запуск
Запуск --> Разогрев
Разогрев --> Ожидание : недостаток ресурса
Разогрев --> Полёт : ресурс имеется
Полёт --> [*]
```

stateDiagram-v2

```
state "This is a state description" as s2
```

This is a state description



Диаграмма состояний. Ветвления и форки. PlantUML.

@startuml Диаграмма состояний

```
state if_state <<choice>>
state "Число положительное" as IsPositive
state "Истина" as True
state "Ложь" as False
```

```
[*] --> IsPositive
IsPositive --> if_state
if_state --> False: if n < 0
if_state --> True : if n >= 0
```

@enduml

@startuml Диаграмма состояний

```
state "Левое состояние" as State1
state "Правое состояние" as State2
state fork_state <<fork>>
state join_state <<join>>
[*] --> fork_state
fork_state --> State1
fork_state --> State2
State1 --> join_state
State2 --> join_state
join_state --> State3
State3 --> [*]
```

@enduml

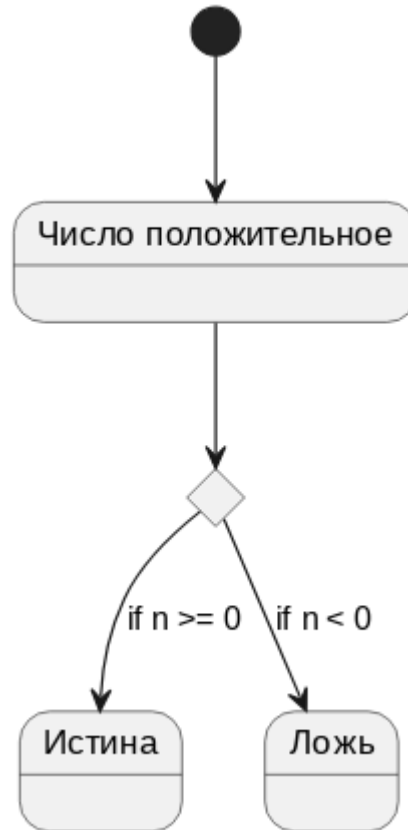


Диаграмма состояний. Ветвления и Форки. Mermaid

stateDiagram-v2

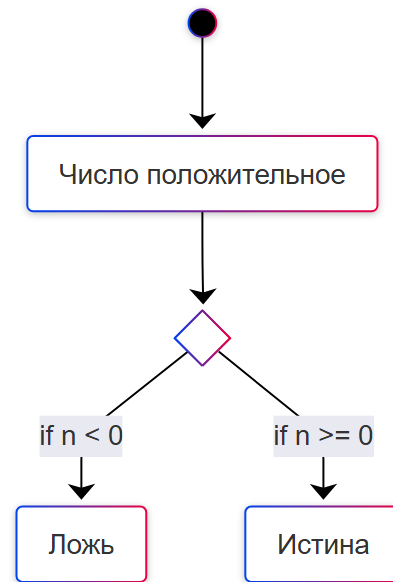
```
state if_state <<choice>>
state "Число положительное" as
IsPositive
state "Истина" as True
state "Ложь" as False
```

```
[*] --> IsPositive
IsPositive --> if_state
if_state --> False: if n < 0
if_state --> True : if n >= 0
```

stateDiagram-v2

```
state "Левое состояние" as State1
state "Правое состояние" as State2
state fork_state <<fork>>
state join_state <<join>>
[*] --> fork_state
fork_state --> State1
fork_state --> State2
State1 --> join_state
State2 --> join_state
join_state --> State3
State3 --> [*]
```

Ветвления



Форки

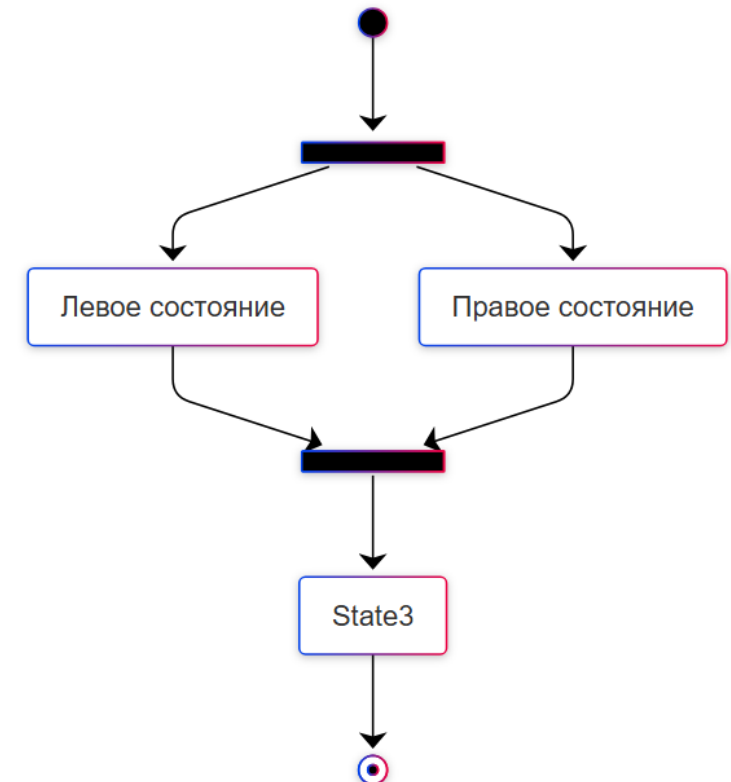


Диаграмма состояний. Конкурентность. PlantUML

@startuml Диаграмма состояний

[*] --> Active

```
state Active {  
    [*] --> NumLockOff  
    NumLockOff --> NumLockOn : EvNumLockPressed  
    NumLockOn --> NumLockOff : EvNumLockPressed  
    --  
    [*] --> CapsLockOff  
    CapsLockOff --> CapsLockOn : EvCapsLockPressed  
    CapsLockOn --> CapsLockOff : EvCapsLockPressed  
    --  
    [*] --> ScrollLockOff  
    ScrollLockOff --> ScrollLockOn : EvScrollLockPressed  
    ScrollLockOn --> ScrollLockOff : EvScrollLockPressed  
}
```

@enduml

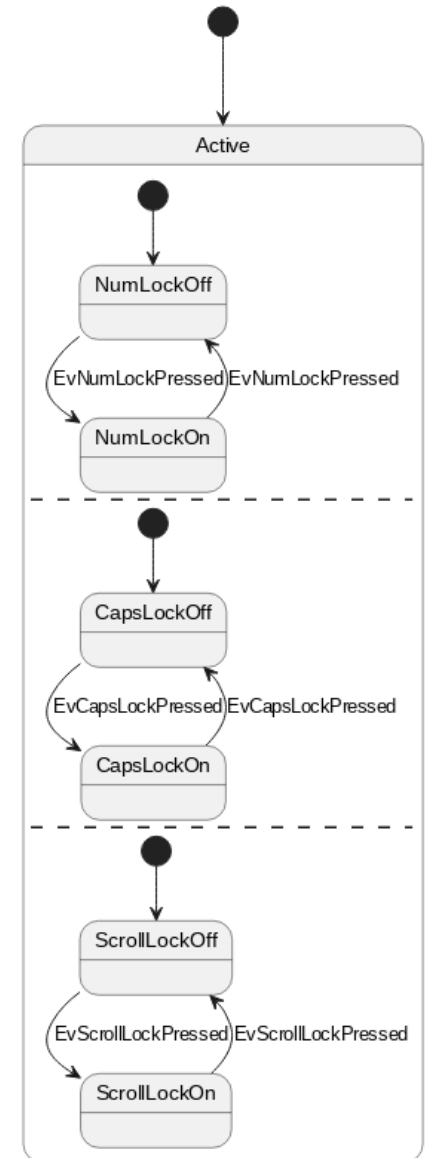


Диаграмма состояний. Конкурентность. Mermaid

stateDiagram-v2

[*] --> Active

state Active {

[*] --> NumLockOff

NumLockOff --> NumLockOn : EvNumLockPressed

NumLockOn --> NumLockOff : EvNumLockPressed

--

[*] --> CapsLockOff

CapsLockOff --> CapsLockOn : EvCapsLockPressed

CapsLockOn --> CapsLockOff : EvCapsLockPressed

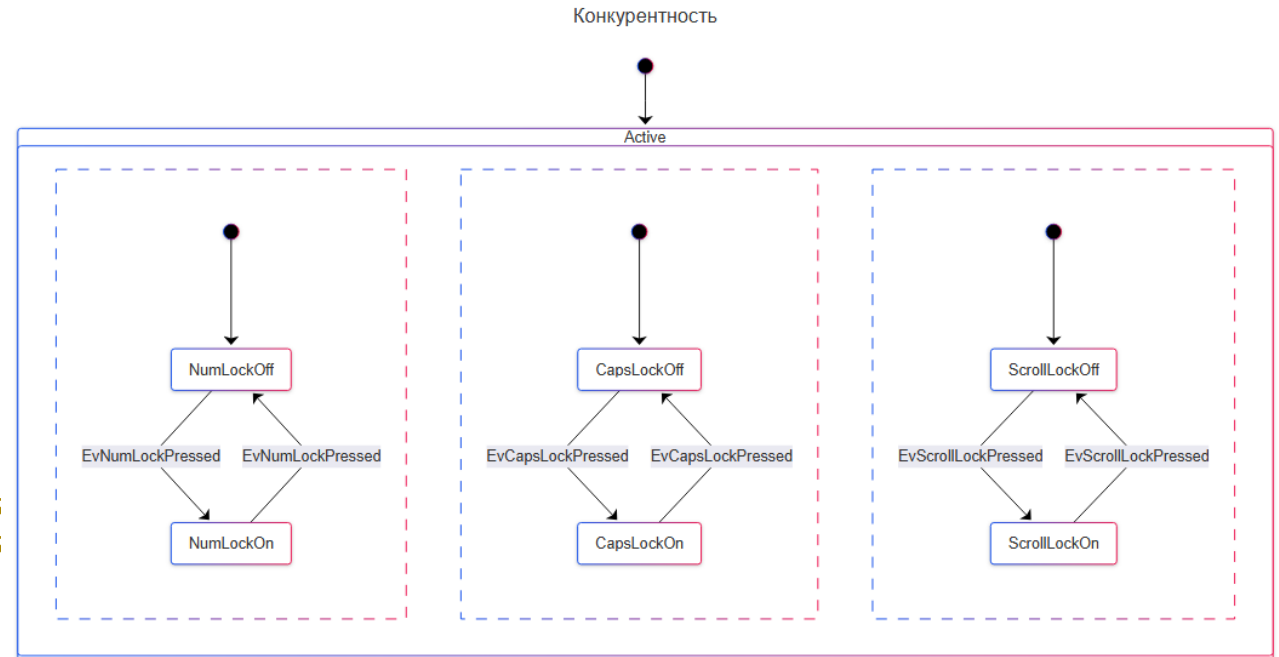
--

[*] --> ScrollLockOff

ScrollLockOff --> ScrollLockOn : EvScrollLockPress

ScrollLockOn --> ScrollLockOff : EvScrollLockPress

}



Диаграммы активности

Диаграмма активности. PlantUML

@startuml

start

:Declare system/loop parameters and set up counter/

:Discretize plant using ZOH method;

repeat

:Set B = Binitial;

:Increment counter by 1;

repeat

:Set K = Kinitial;

:Close position feedback loop;

:Resample system using Tustin method;

:Close velocity feedback loop and obtain system

characteristic equation;

backward :Decrement K by Kinc;

repeat while (Is stability criterion satisfied?) is

(no) not (yes)

:Save K/

:Decrement K by Kinc;

repeat while (Is B = Bfinal?) is (no) not (yes)

stop

@enduml

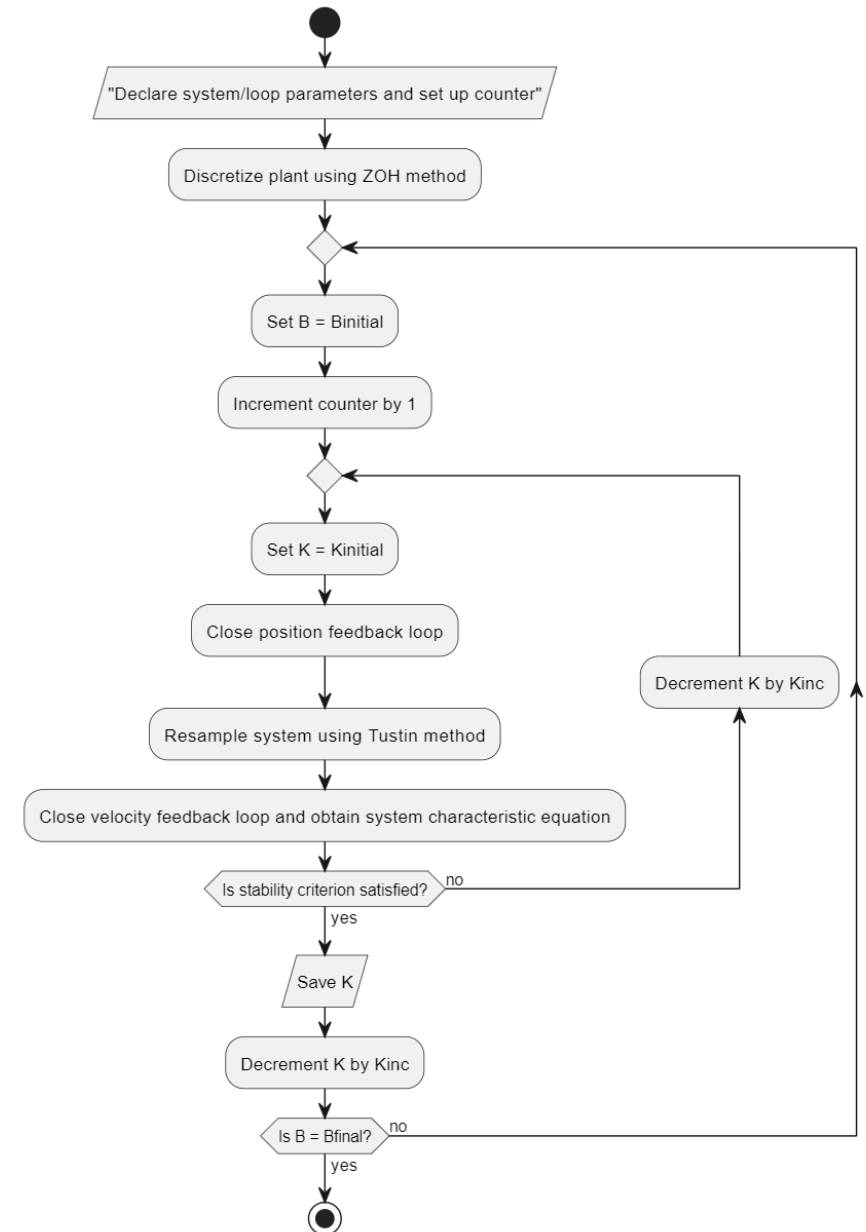
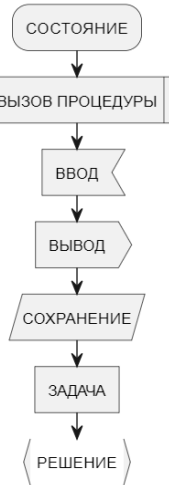


Диаграмма активности. Блоки. PlantUML

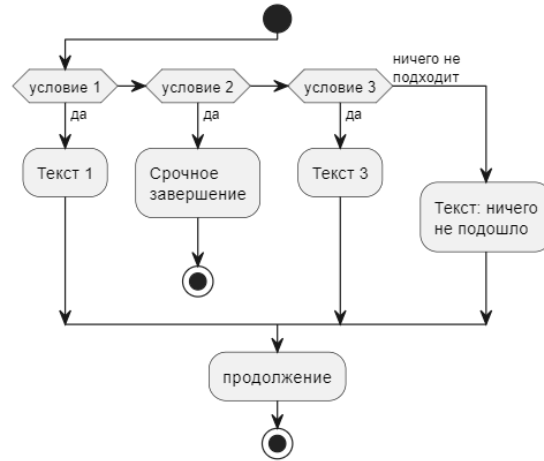
Блоки

```
@startuml
:СОСТОЯНИЕ;
:ВЫЗОВ ПРОЦЕДУРЫ|
:ВВОД<
:ВЫВОД>
:СОХРАНЕНИЕ/
:ЗАДАЧА]
:РЕШЕНИЕ}
@enduml
```

- ;- STATE, СОСТОЯНИЕ
- | - CALL, ВЫЗОВ ПРОЦЕДУРЫ
- <- INPUT, ВВОД
- >- OUTPUT, ВЫВОД
- /- SAVE, СОХРАНЕНИЕ
-]- TASK, ЗАДАЧА
- }- DECISION, РЕШЕНИЕ



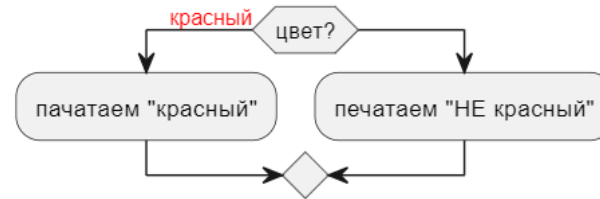
Условия. Простые



```
@startuml
start
if (условие 1) then (да)
:Текст 1;
elseif (условие 2) then (да)
:Срочное\завершение;
stop
elseif (условие 3) then (да)
:Текст 3;
else (ничего не\подоходит)
:Текст: ничего\не подошло;
endif

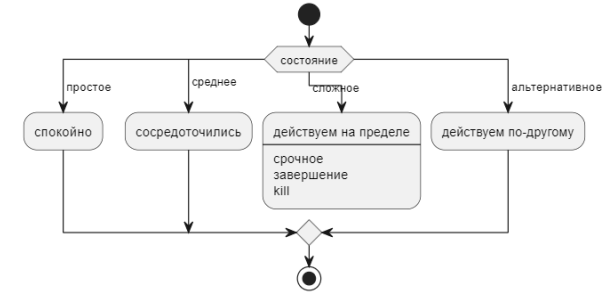
:продолжение;

stop
@enduml
```



```
@startuml
if (цвет?) is (<color:red>красный)
then
:печатаем "красный";
else
:печатаем "НЕ красный";
endif
@enduml
```

Условия. Множественные



```
@startuml
start
switch ( состояние )
case ( простое )
:спокойно;
case ( среднее )
:сосредоточились;
case ( сложное )
:действуем на пределе
----
срочное
завершение
kill;
kill
case ( альтернативное )
:действуем по-другому;
endswitch

stop
@enduml
```

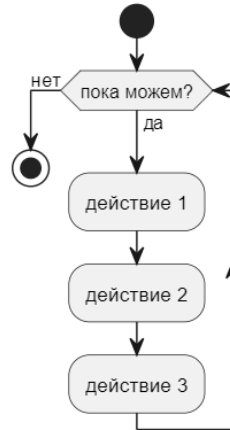
Диаграмма активности. Циклы. PlantUML

С постусловием



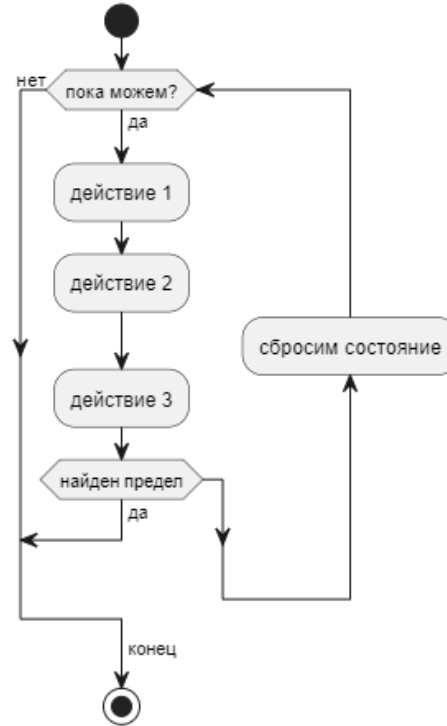
```
@startuml
start
repeat
    :действие 1;
    :действие 2;
    :действие 3;
repeat while
(повторим?) is (да) not
(нет)
stop
@enduml
```

С предусловием



```
@startuml
start
while (пока можем?) is (да)
    :действие 1;
    :действие 2;
    :действие 3;
endwhile (нет)
stop
@enduml
```

Досрочный выход

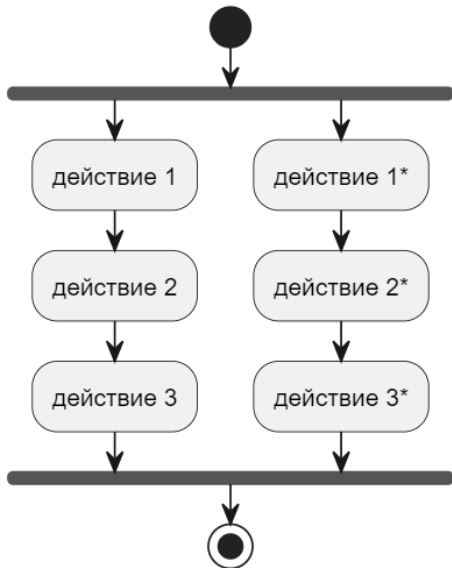


```
@startuml
start
while (пока можем?) is (да)
    :действие 1;
    :действие 2;
    :действие 3;
    if (найден предел) then (да)
        break
    endif
    backward :сбросим состояние;
endwhile (нет)
->конец;
stop
@enduml
```


Диаграмма активности. Форки. PlantUML

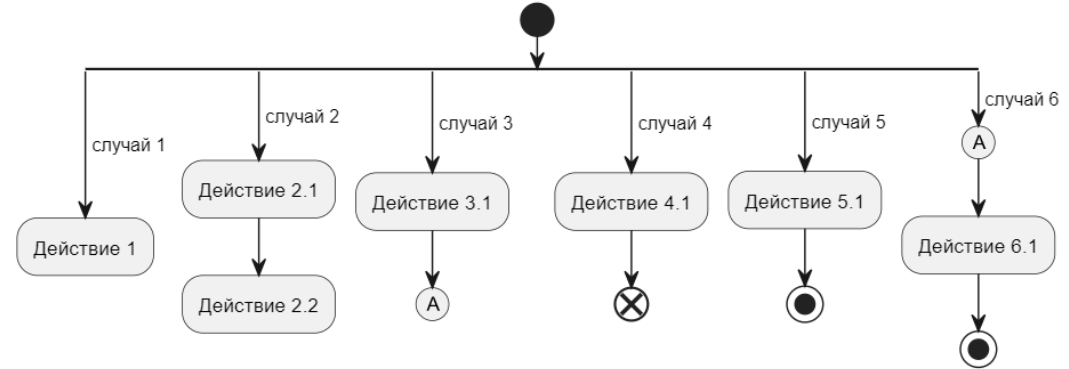
Форки (параллелизм)

```
@startuml
start
fork
:действие 1;
:действие 2;
:действие 3;
fork again
:действие 1*;
:действие 2*;
:действие 3*;
end fork
stop
@enduml
```



Разделение

```
@startuml
start
split
->случай 1;
:Действие 1;
kill
split again
->случай 2;
:Действие 2.1;
:Действие 2.2;
detach
split again
->случай 3;
:Действие 3.1;
(A)
detach
split again
->случай 4;
:Действие 4.1;
end
split again
->случай 5;
:Действие 5.1;
stop
split again
->случай 6;
(A)
:Действие 6.1;
stop
end split
@enduml
```



Однопоток/многопоток

```
@startuml
start
if (многопоточность?) then (да)
fork
:Операция 1;
fork again
:Операция 2;
end fork
else (всё в одном\потоке)
:Операция 1;
:Операция 2;
endif
stop
@enduml
```

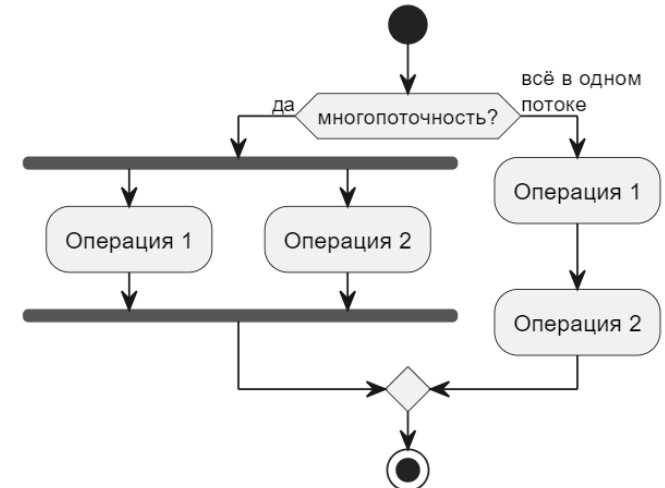


Диаграмма активности. Mermaid

```
config:
  layout: dagre
---
```

flowchart TB

```
S1(["START"]) --> DE[/Declare system/loop parameters and
set up counter/]
DE --> DI("Discretize plant using ZOH method")
DI --> SB("Set B = Binitial")
SB --> IC("Increment counter by 1")
IC --> SK("Set K = Kinitial")
SK --> CP("Close position feedback loop")
CP --> RS("Resample system using Tustin method")
RS --> CV("Close velocity feedback loop and obtain system
characteristic equation")
CV --> IFS{"Is stability criterion satisfied?"}
IFS -- Yes --> SVK[/Save K/]
IFS -- No --> DK("Decrement K by Kinc")
SVK --> BKL("Break K loop")
DK --> IBB{"Is B = Bfinal?"}
BBK --> IBB
IBB -- No --> SB
IBB -- Yes --> End(["End"])
```

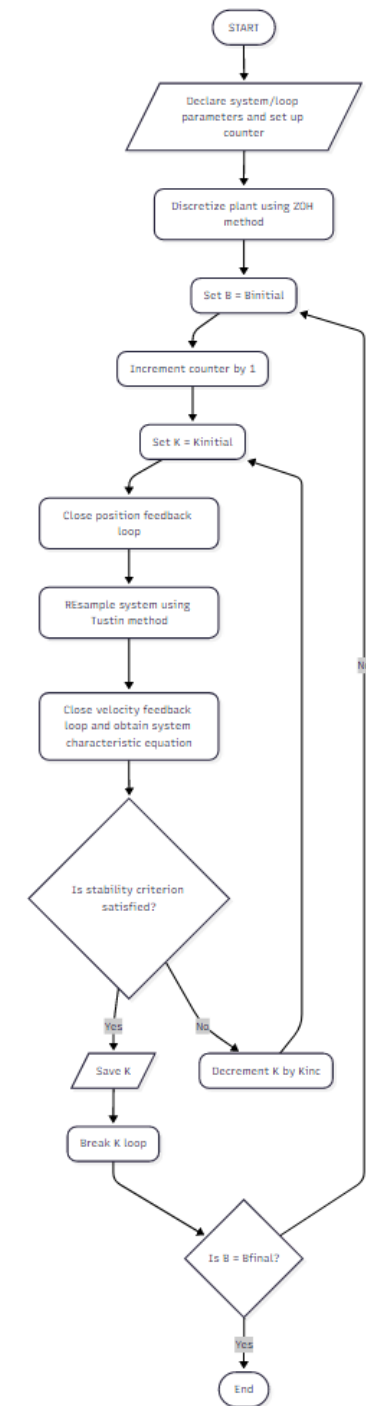


Диаграмма активности. Блоки. Mermaid

```

---
config:
  layout: elk
---
flowchart LR
  subgraph s1["Shapes"]
    direction TB
    n9["Untitled Node"]
    n8(("Circle"))
    n10["Triangle"]
    n11["Trapezoid"]
    n12["Trapezoid
Reversed"]
    n13["Hexagon"]
    n14(["Rounded"])
    n15{"Diamond"}
    n16{{"Hexagon"}}
    n17[("Database")]
    n18[/"Parallelogram"/]
    n19[/"Trapezoid"\]
  end
  subgraph s2["Flowchart"]
    direction TB
    n1["Standard Process"]
    n2["Event"]
    n3{"Decision"}
    n4[/"Out In"/]
    n5[/"In Out"/]
    n6["Loop Limit"]
    n7["Sub Process"]
  end
end
n9 --> n8
n10 --> n11
n11 --> n12
n12 --> n13
n13 --> n14
n14 --> n15
n15 --> n16
n16 --> n17
n17 --> n18
n18 --> n19
n10@{ shape: tri}
n11@{ shape: trap-b}
n12@{ shape: trap-t}
n13@{ shape: hex}
n1@{ shape: proc}
n2@{ shape: event}
n3@{ shape: decision}
n4@{ shape: out-in}
n5@{ shape: in-out}
n6@{ shape: loop-limit}
n7@{ shape: subproc}

```

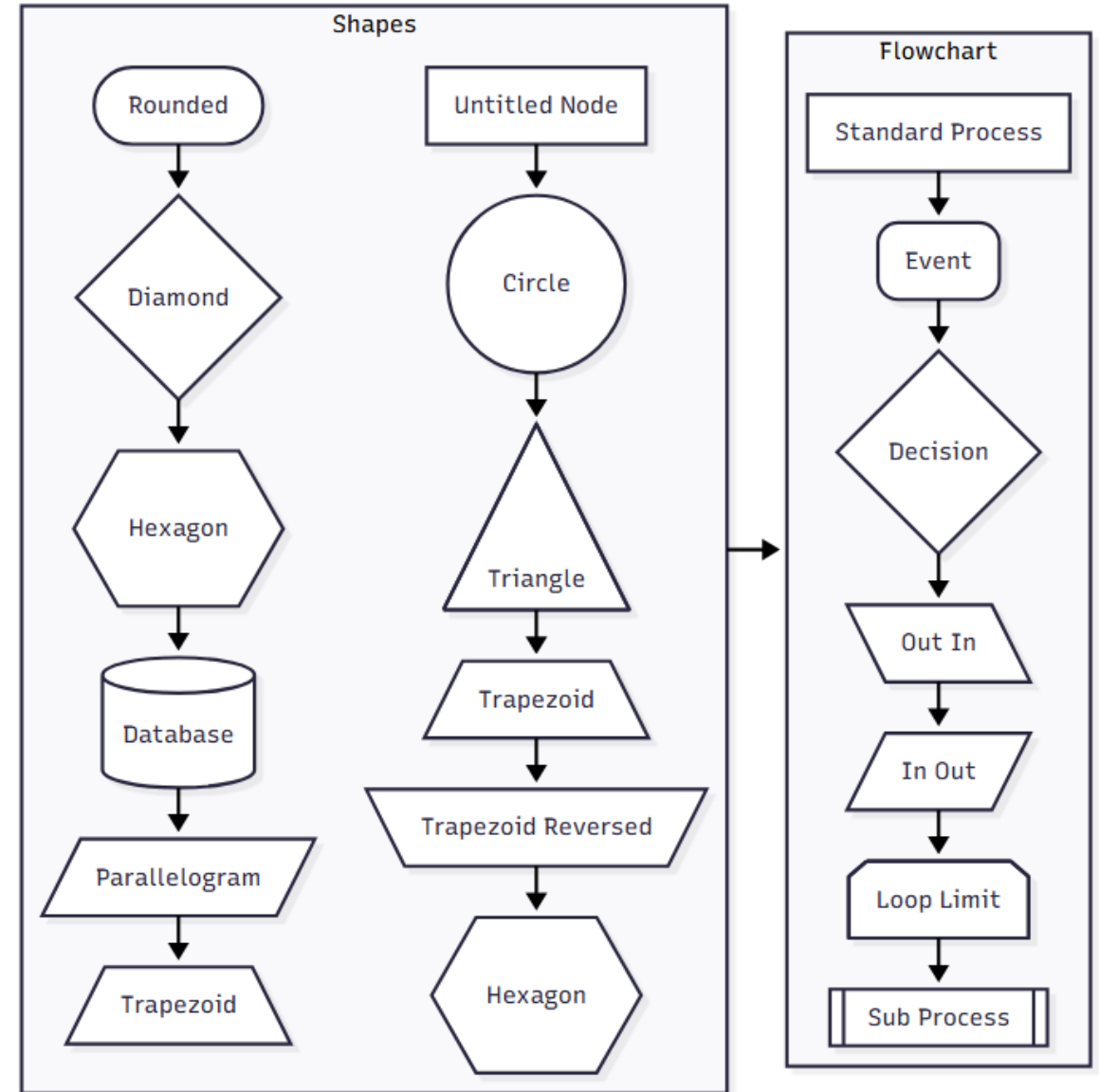
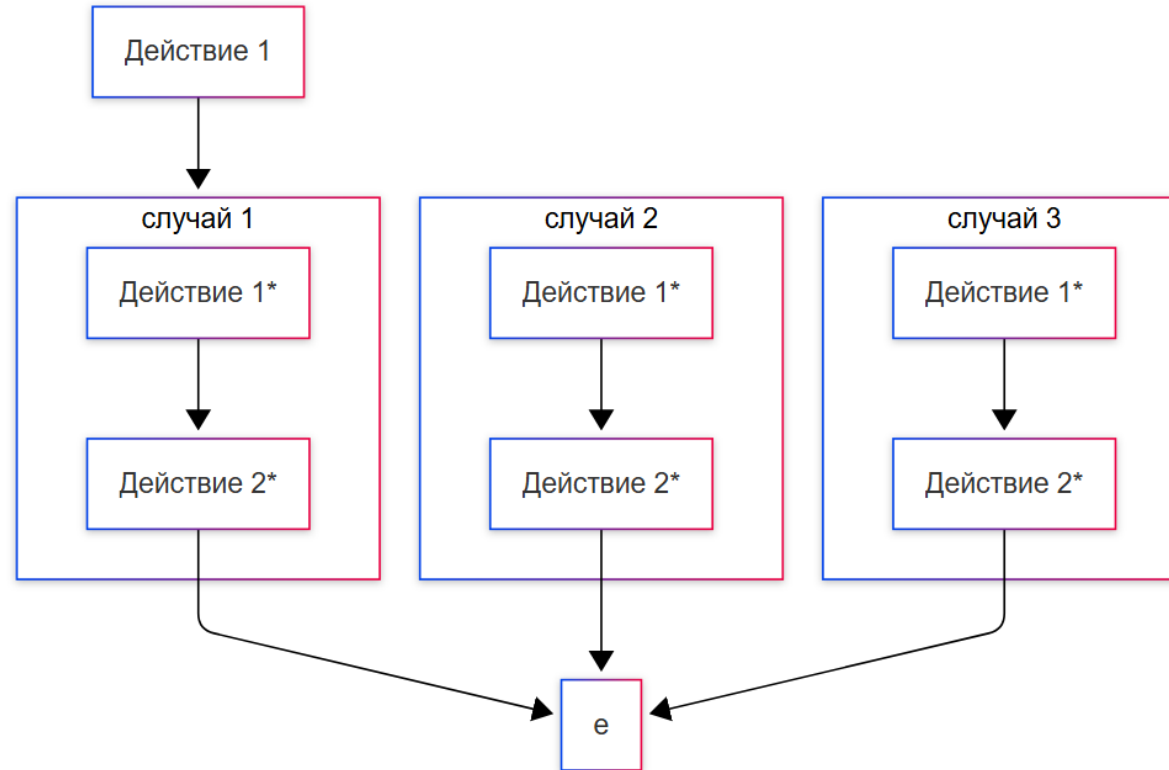


Диаграмма активности. Разделы. Mermaid

flowchart TB

```
a1[Действие 1]-->s1
subgraph s1[случай 1]
a1*[Действие 1*]-->a2*
a2*[Действие 2*]
end
subgraph случай 2
a1+[Действие 1*]-->a2+
a2+[Действие 2*]
end
subgraph случай 3
a1-[Действие 1*]-->a2-
a2-[Действие 2*]
end
a2* --> e
a2+ --> e
a2- --> e
```



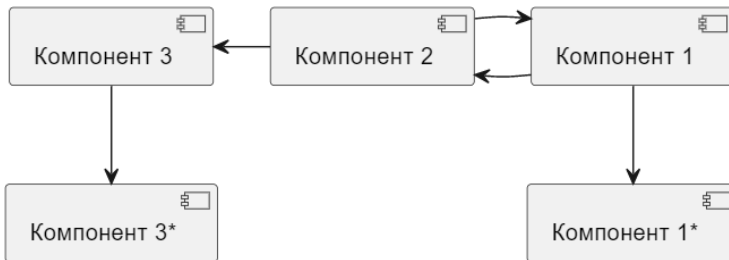
Диаграммы КОМПОНЕНТОВ

Диаграмма компонентов. PlantUML

Форки (параллелизм)

```
@startuml
```

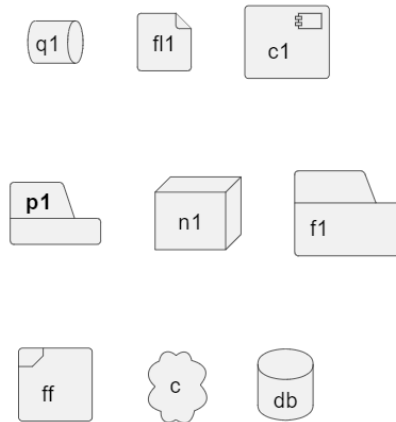
```
[Компонент 1] -left-> [Компонент 2]
[Компонент 2] -left-> [Компонент 3]
[Компонент 2] -right-> [Компонент 1]
[Компонент 3] -down-> [Компонент 3*]
```



Виды блоков

```
@startuml
```

```
queue q1
file fl1
component c1
package p1
node n1
folder f1
frame ff
cloud c
database db
@enduml
```



Пакеты (группы)

```
@startuml
```

```
package p1 {
    component p1c1
    component p1c2
}
```

```
package p2 {
    component p2c1
    component p2c2
    component p2c3
}
```

```
p2c1 -right-> p2c2
p2c2 -down-> p2c3
```

```
}
```

```
package p3 {
    database db1
    database db2
}
```

```
p1c1 ..> db1
p2c1 ..> db2
```

```
@enduml
```

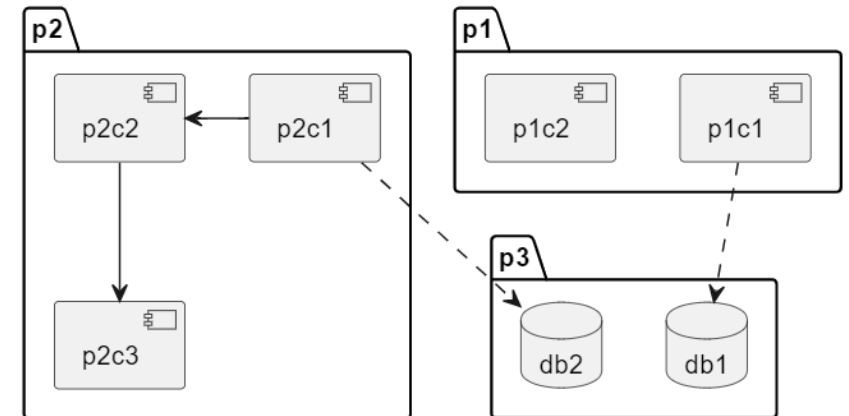


Диаграмма компонентов.Mermaid

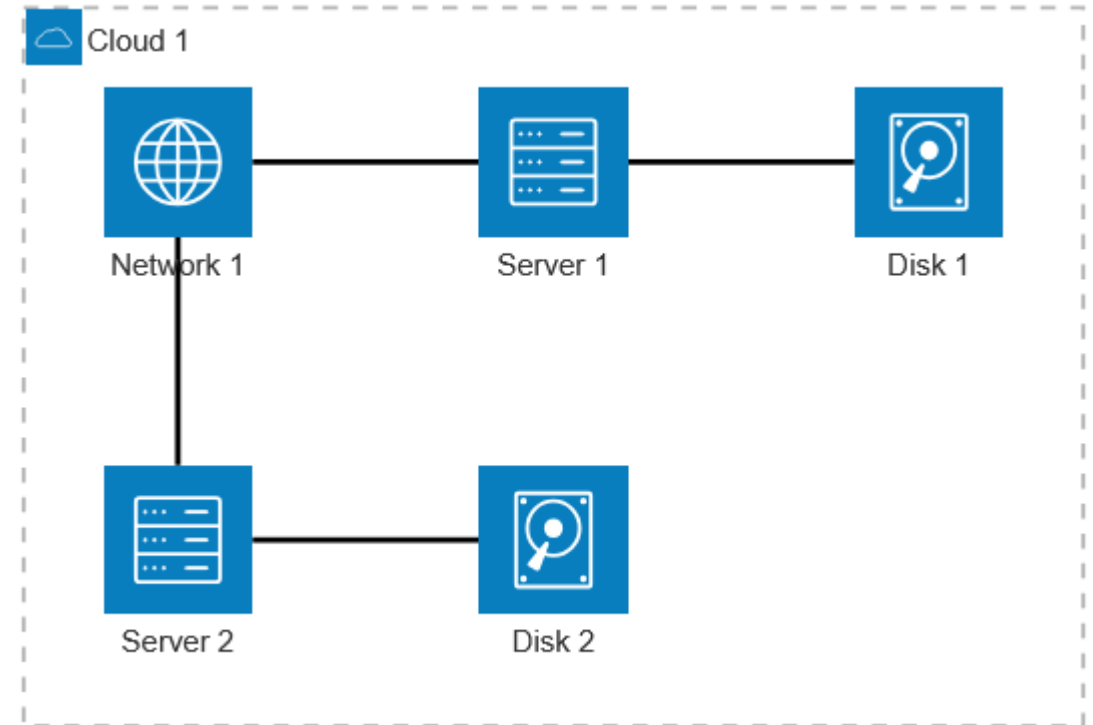
В Mermaid используется подтип Архитектурная диаграмма, имеются ограничения

architecture-beta

```
group cloud1(cloud)[Cloud 1]
```

```
service server1(server)[Server 1] in cloud1
service server2(server)[Server 2] in cloud1
service disk2(disk)[Disk 2] in cloud1
service disk1(disk)[Disk 1] in cloud1
service net1(internet)[Network 1] in cloud1
```

```
disk1:L -- R:server1
disk2:L -- R:server2
server1:L -- R:net1
net1:B -- T:server2
```



Синтаксис:

```
service {service id}({icon name})[{title}] (in {parent id})?
```

Диаграммы развёртывания

Диаграмма развёртывания. Элементы. PlantUML

```
@startuml
action action
agent agent
artifact artifact
boundary boundary
card card
cloud cloud
collections collections
component component
control control
database database
entity entity
file file
folder folder
frame frame
hexagon hexagon
interface interface
label label
node node
package package
person person
process process
queue queue
rectangle rectangle
stack stack
storage storage
@enduml
```

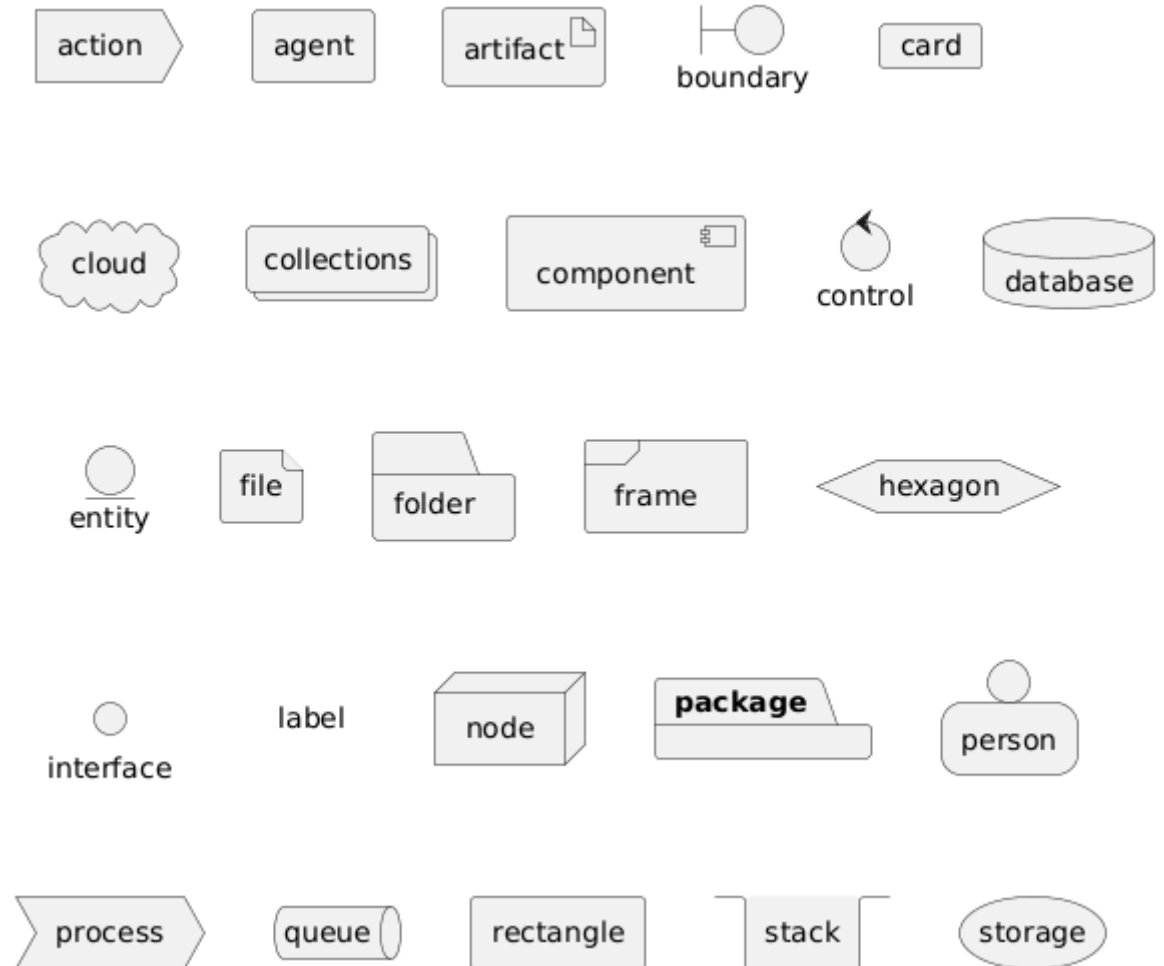
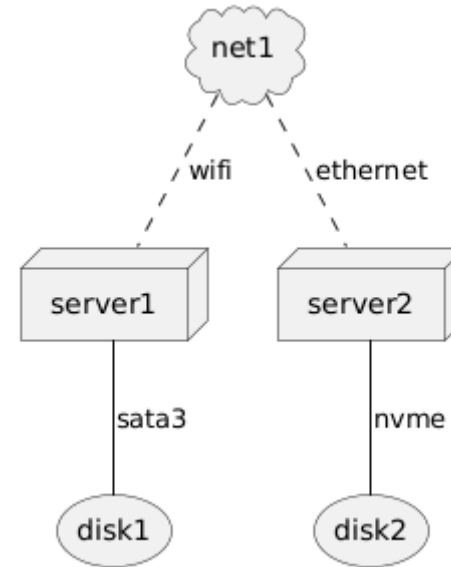


Диаграмма развёртывания. Связи. PlantUML

```
@startuml
```

```
node server1
node server2
storage disk1
storage disk2
cloud net1
server1 -- disk1 : sata3
server2 -- disk2 : nvme
net1 .. server1 : wifi
net1 .. server2 : ethernet
```

```
@enduml
```



Диаграммы последовательности

Диаграмма последовательности. Элементы. PlantUML

@startuml

```
participant База as Foo
actor Актор as Foo1
boundary Граница as Foo2
control Управление as Foo3
entity Сущность as Foo4
database БД as Foo5
collections Коллекции as Foo6
queue Очередь as Foo7

Foo -> Foo1 : To actor
Foo -> Foo2 : To boundary
Foo -> Foo3 : To control
Foo -> Foo4 : To entity
Foo -> Foo5 : To database
Foo -> Foo6 : To collections
Foo -> Foo7 : To queue
```

@enduml

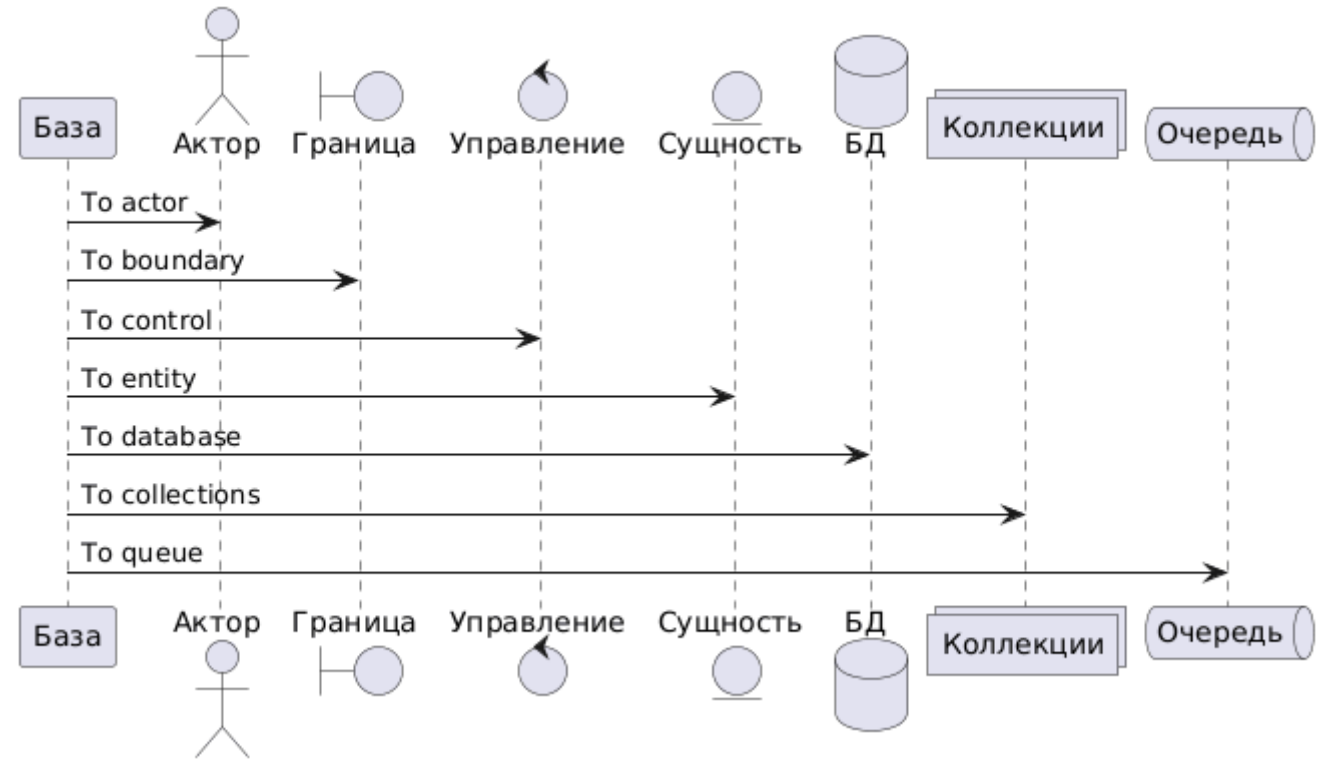


Диаграмма последовательности. Альтернативы и время жизни. PlantUML

```
@startuml
actor Alice
control Log
Alice -> Bob: Запросить ресурс
Bob -> Alice: Вернуть ресурс
group My own label [My own label 2]
  Alice -> Log : Записать в лог
  activate Log
  alt
    Alice --> Tom : послать сигнал
    activate Tom
    deactivate Tom
  else
    loop 1000 раз
      Bob -> Tom: отправить пакеты
      activate Tom
    end
  end
  deactivate Tom
  Alice -> Log : Записать в лог
  deactivate Log
end
@enduml
```

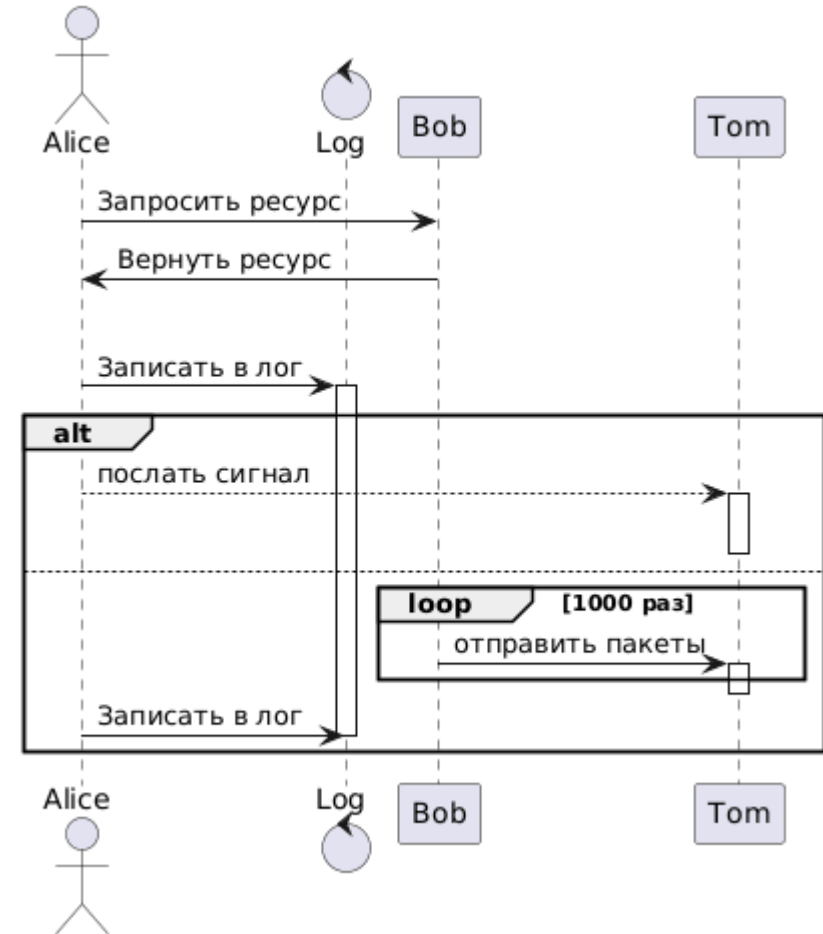


Диаграмма последовательности. Альтернативы и время жизни. Mermaid

```
sequenceDiagram
actor Alice
participant Log
```

```
Alice->>Bob: Запросить ресурс
```

```
Bob->>Alice: Вернуть ресурс
```

```
Alice->>Log : Записать в лог
```

```
activate Log
```

```
alt
```

```
    Alice->>Tom : послать сигнал
```

```
    activate Tom
```

```
    deactivate Tom
```

```
else
```

```
loop 1000 раз
```

```
    Bob->>Tom: отправить пакеты
```

```
    activate Tom
```

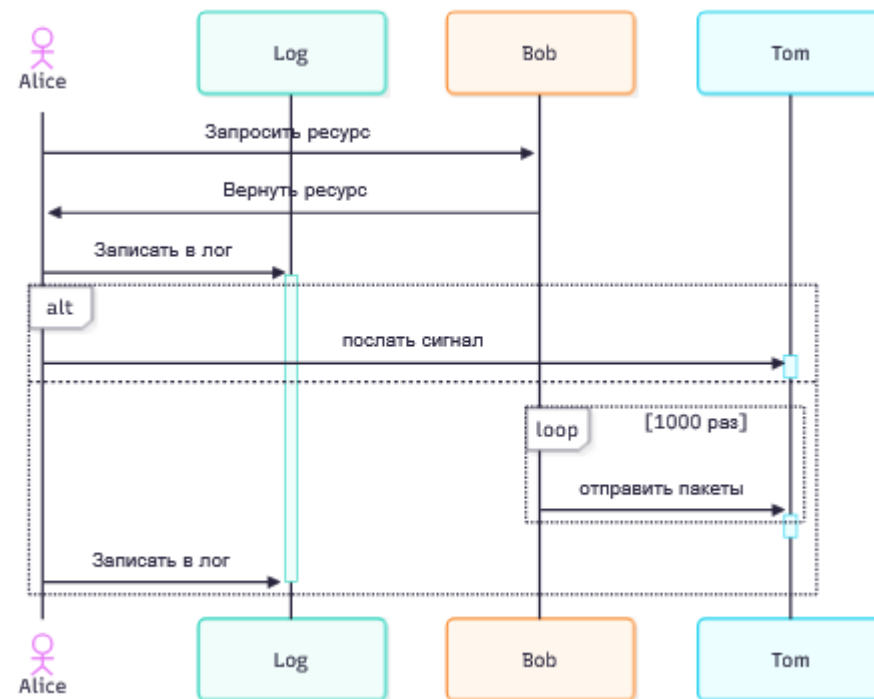
```
end
```

```
deactivate Tom
```

```
Alice->>Log : Записать в лог
```

```
deactivate Log
```

```
end
```



Ссылки

- <https://plantuml.com/ru/>
- <https://mermaid.js.org/intro/>
- https://www.mermaidchart.com/play?utm_source=mermaid_live_editor&utm_medium=toggle