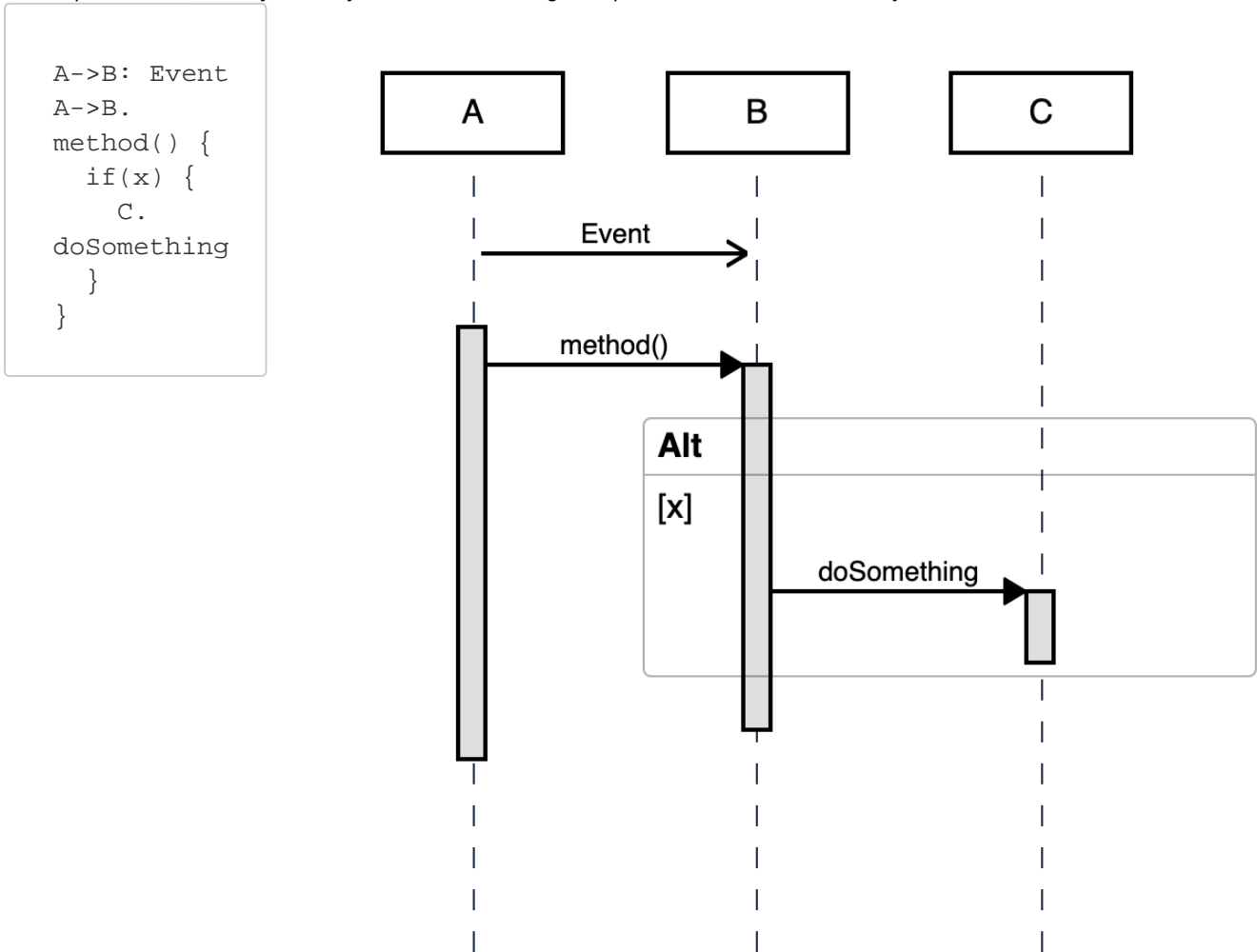


Sequence diagram syntax

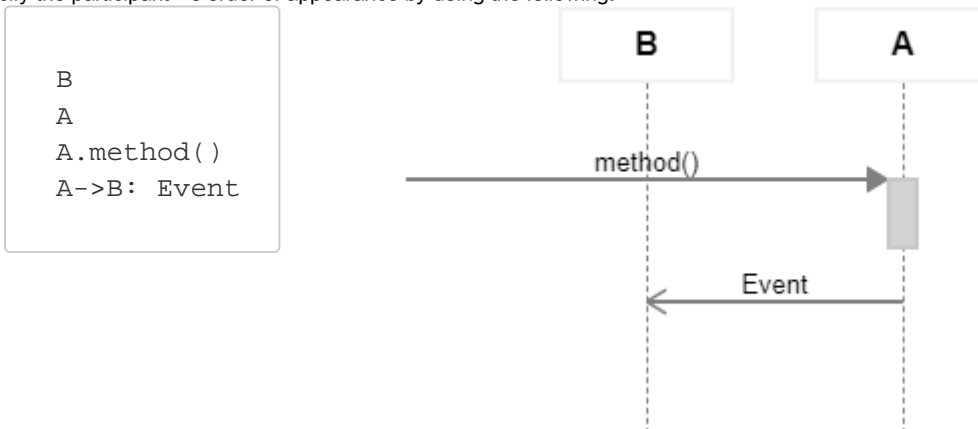
A sequence diagram is an interaction diagram that shows how processes operate with one another and in what order.

Most developers would find the syntax fairly familiar. The following example demonstrates some basic syntaxes.



Participants

The participants can be defined implicitly as in the first example on this page. The participants are rendered in order of appearance in the diagram source text. Sometimes you might want to show the participants in a different order than how they appear in the first message. It is possible to specify the participant's order of appearance by doing the following:



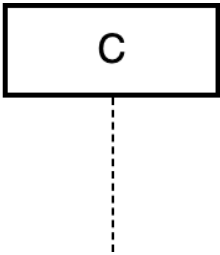
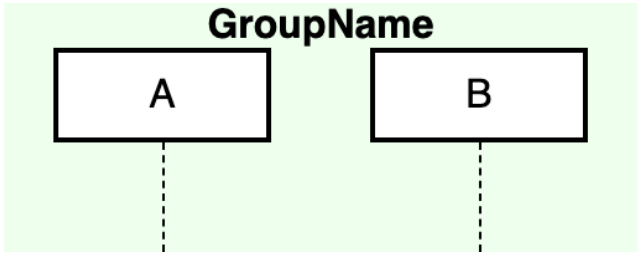
Participant group

We can group the participants with `group` keyword.

```

group
GroupName {
  A
  B
}
C

```



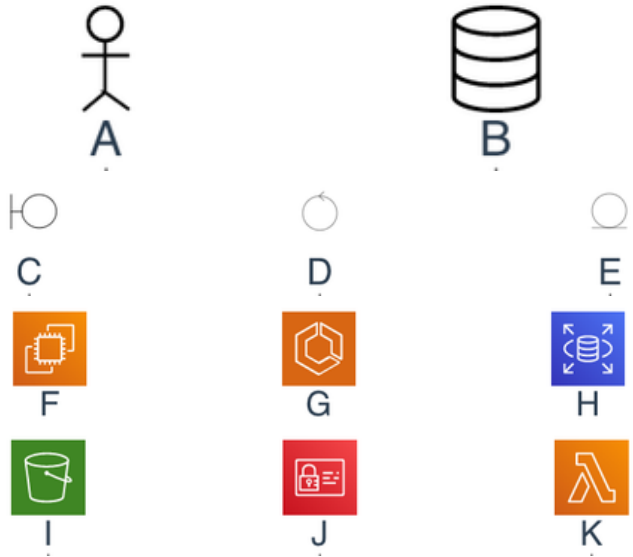
Participant type (Annotation)

We can change the shape of the participant representation with annotations.

```

@Actor A
@Database B
@Boundary C
@Control D
@Entity E
@EC2 F
@ECS G
@RDS H
@S3 I
@IAM J
@Lambda K

```



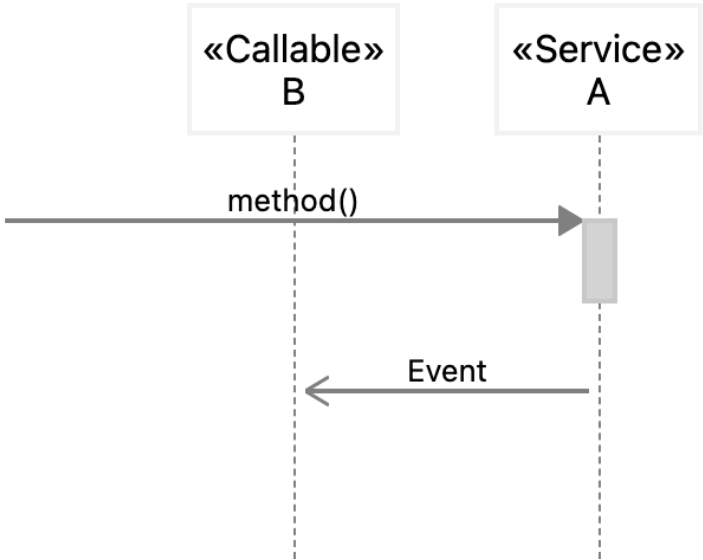
Stereotype

It is possible to add stereotypes to participants using << and >>.

```

<<Callable>>
B
<<Service>>
A
A.method()
A->B: Event

```



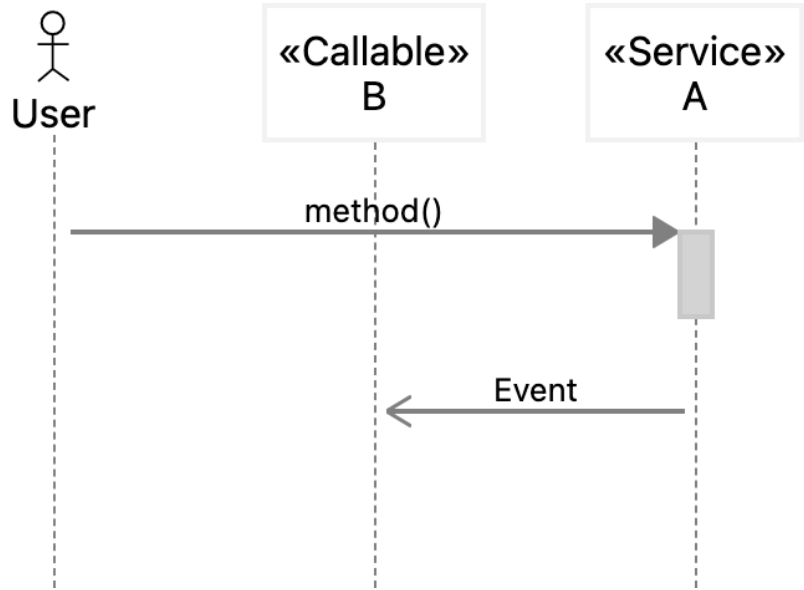
Starter

ADVANCED

By default, the “client” of the interaction is not shown in the diagram. However, you can specify a “client” with the `@Starter` keyword. Specifically, if the starter’s name is “User” or “Actor”, we will use a Stickman icon. `@Starter` must be put after you have declared all participants and before any messages.

```

<<Callable>>
B
<<Service>>
A
@Starter
(User)
A.method()
A->B: Event
    
```

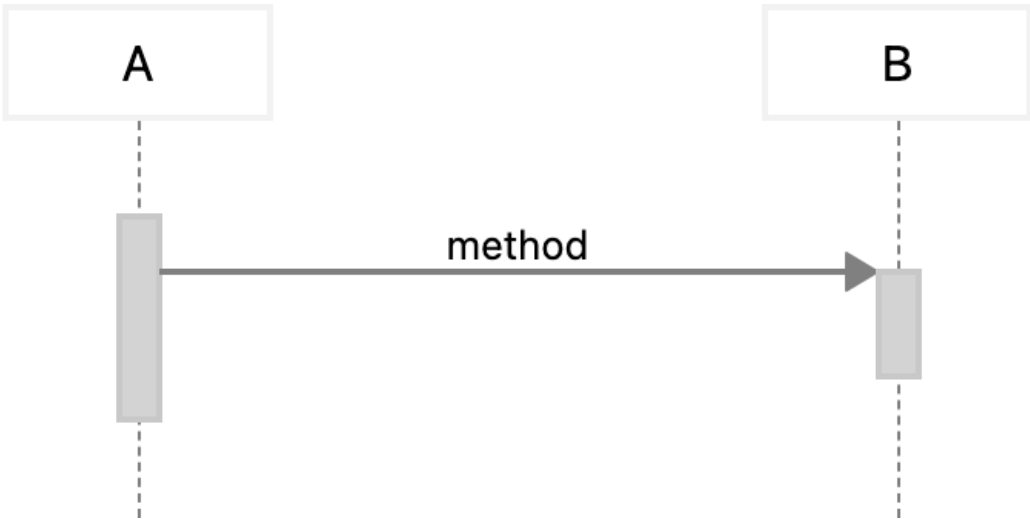
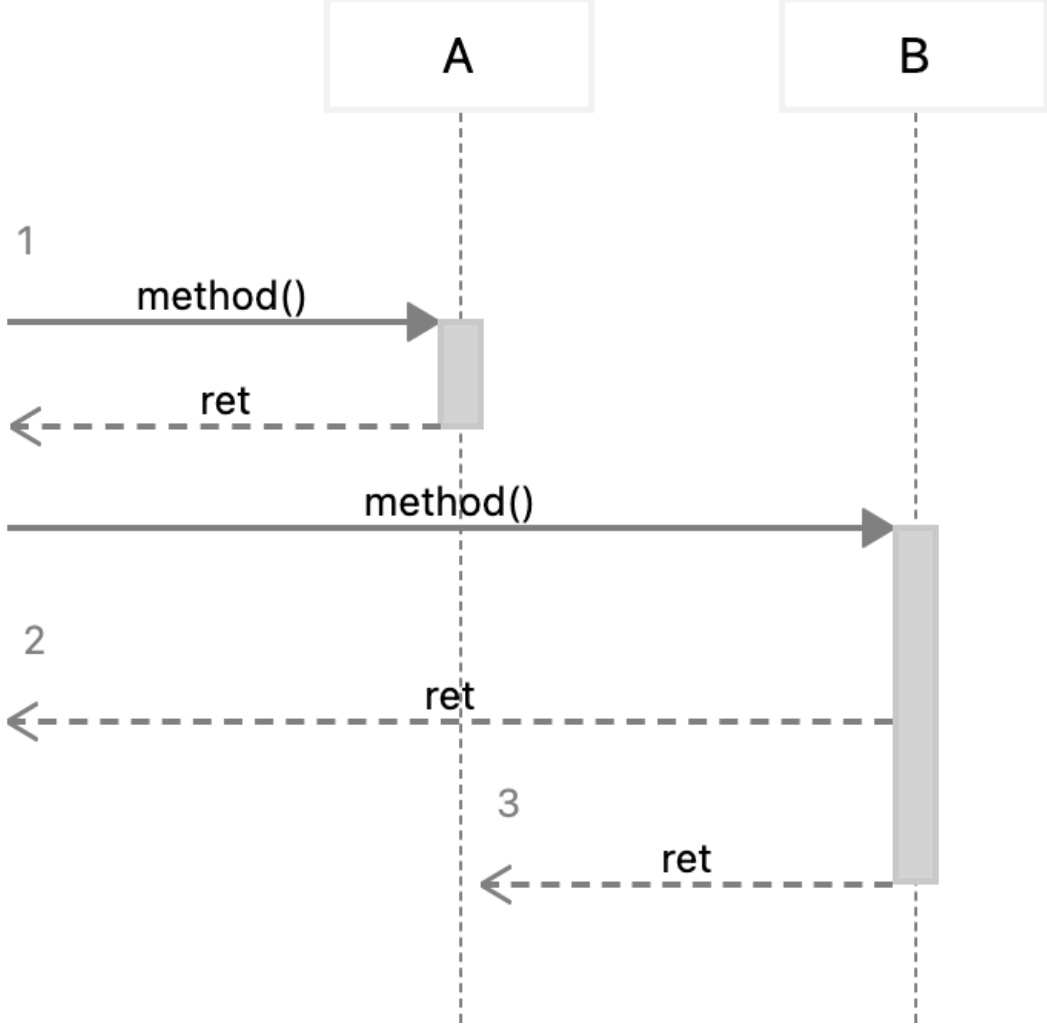


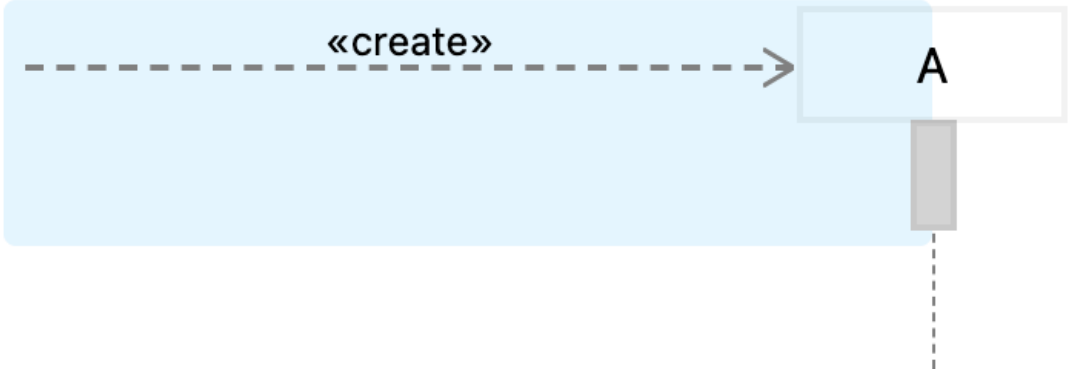
Messages

A message is shown as a line from the sender MessageEnd to the receiver MessageEnd.

See Unified Modeling Language v2.5.1, section 17.4.4.1.

Message type	DSL	Line and arrow head (Spec)	
Async	A->B: Async message	solid line with open arrowhead	<p>The diagram shows two lifelines, A and B, represented by boxes. A solid arrow labeled 'Async Message' points from the lifeline of A to the lifeline of B. The arrow has an open arrowhead.</p>

Sync	A. method()	filed arrowhead	 <pre> sequenceDiagram participant A participant B A->>B: method </pre>
Reply	<ol style="list-style-type: none"> return = A. method return @return 	dashed line with either an open or filled arrowhead * Zen UML render use open arrowhead.	 <pre> sequenceDiagram participant A participant B A->>B: 1 method() B-->>A: 1 ret A->>B: 2 method() B-->>A: 2 ret B->>A: 3 method() A-->>B: 3 ret </pre>

Object creation	new ClassName()	dashed line with an open arrowhead	
Object deletion	NOT SUPPORTED YET	Must end in a Destruction Occurrence Specification	
Lost	NOT SUPPORTED YET	A small black circle at the arrow end of the message	
Found	NOT SUPPORTED YET	A small black circle at the starting end of the message	

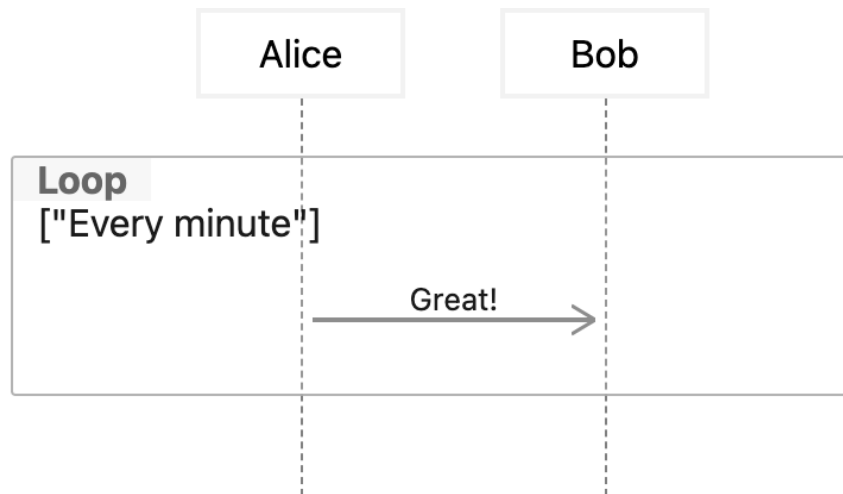
Loops

The loop operand will be repeated a number of times. This is expressed by the notation:

```
while(condition) {}  
for(enumerator) {}  
forEach(enumerator) {}
```

See the example below:

```
loop("Every  
minute") {  
  Alice-  
>Bob: Great!  
}
```



Alt

The alt operand represents a choice of behavior. At most one of the operands will be chosen. This is expressed by the notions:

```
if (condition1) {  
  ...  
} else if (condition2) {  
  ...  
} else {  
  ...  
}
```

```
if (x) {  
  A.m1()  
} else if (y) {  
  A.m2()  
} else {  
  A.m3()  
}
```

A

