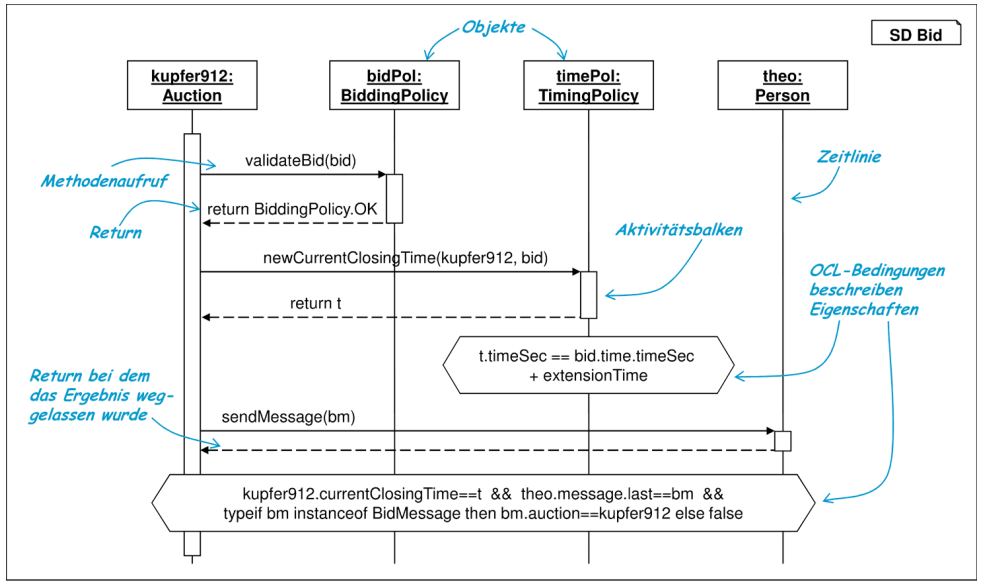
SD-Language

Sequence Diagram Language based on MontiCore

Christian Broering
Brian Sinkovec
Christian Volkmann

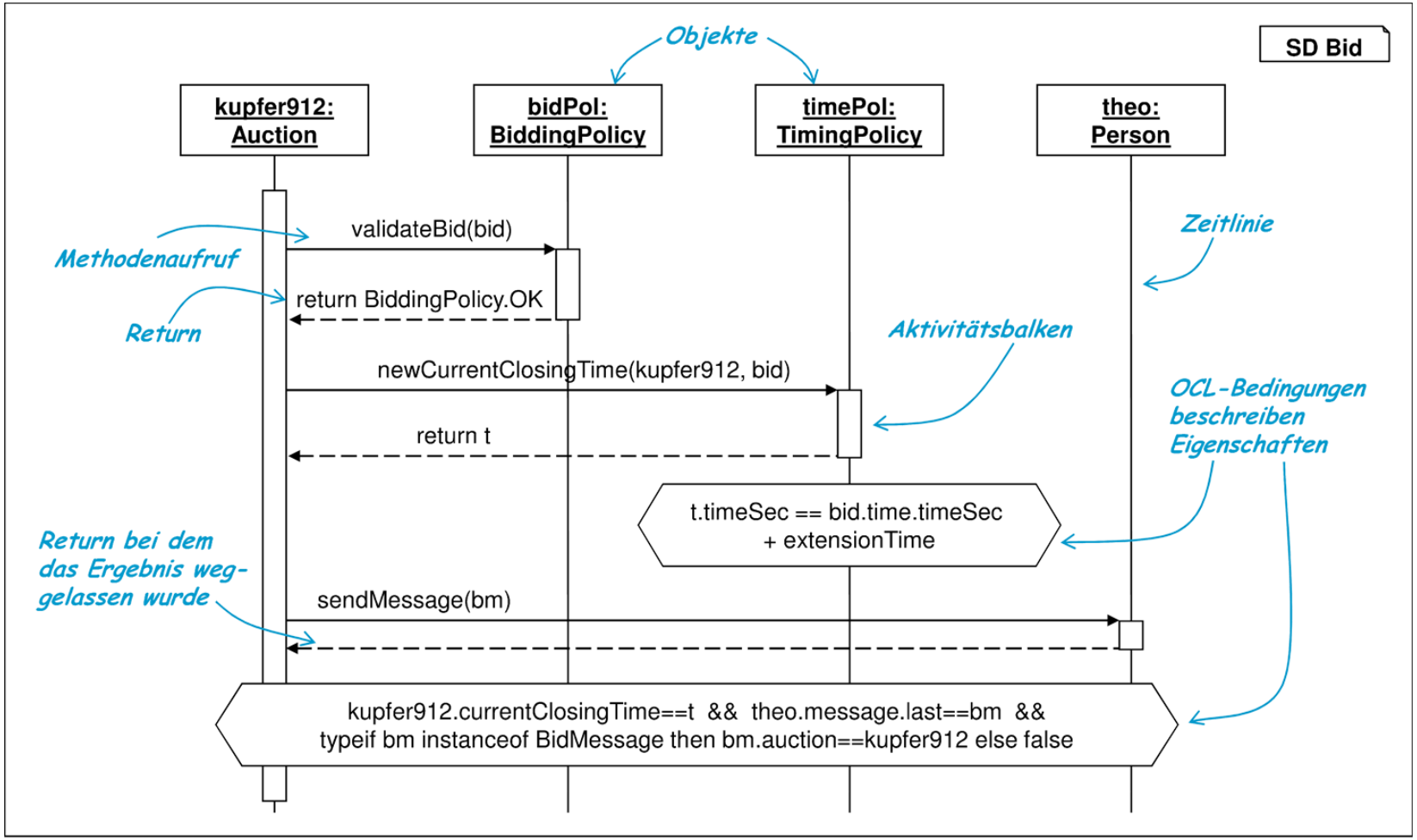
16.07.2020



SE Software Engineering | RWTH AACHEN UNIVERSITY

Sequence Diagram Overview

Sequence what?



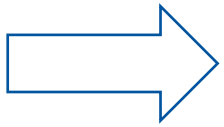
* taken from **B.Rumpe : Modellierung mit UML (**http://mbse.se-rwth.de/book1/)

Grammar

SequenceDiagram, SDBody, SDElement

MC

```
1  symbol scope SequenceDiagram =
2    Stereotype? SDModifier*
3      “sequencediagram” Name “{“
4        SDObject*
5        SDBody
6      “}”
7  ;
8
9  scope SDBody = SDElement*;
10
11 interface SDElement;
12
13 interface SDModifier;
```



SD

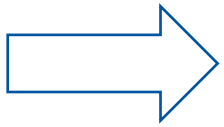
```
1  complete sequencediagram MySD {
2
3    // SD objects ...
4
5    // SD body ...
6
7  }
```


Christian Broering, Brian Sinkovec, Christian Volkmann

SDObject

MC

```
1  SDObject implements Variable =
2    Stereotype? SDModifier*
3    Name (“:” MCObjectType)? “;”
4  ;
```



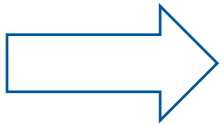
SD

```
1  // ...
2
3  myObj : MyType;
4
5  yourObj;
6
7  visible objWithMod;
8
9  // ...
```

SDInteraction, SDSendMessage

MC

```
1  interface SDInteraction
2    extends SDElement;
3
4  SDSendMessage
5    implements SDInteraction  =
6      SDSource? “->” SDTarget? “:”
7      SDAction (“;” | SDActivityBar)
8  ;
9
10 interface SDSource;
11 interface SDTarget;
12
13 interface SDAction;
14
15 SDActivityBar = “{“ SDElement* “}”;
```



SD

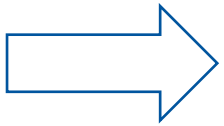
```
1  // ...
2
3  a -> b : foo();
4
5  a -> b : bar() {
6    b -> c : foobar();
7  }
8
9  // ...
```


Christian Broering, Brian Sinkovec, Christian Volkmann

SDNew, SDReturn, SDThrow

MC

```
1  SDNew implements
2    SDInteraction, Variable =
3      SDSource? “->”
4      declarationType:MCObjectType Name
5      “=“ “new”
6      initializationType:MCObjectType
7      Arguments (“;” | SDActivityBar)
8  ;
9
10 SDReturn implements SDAction =
11   (“return” Expression?)?;
12
13 SDThrow implements SDAction =
14   “throw” MCObjectType Arguments?;
```



SD

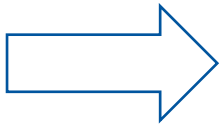
```
1  // ...
2
3  a -> B b = new B();
4
5  a <- b : return;
6
7  a <- c : throw Exception();
8
9  // ...
```


Christian Broering, Brian Sinkovec, Christian Volkmann

SDIncompleteExpression, SDCondition, SDVariableDeclaration

MC

```
1  SDIncompleteExpression implements
2    Expression = “...”;
3
4  SDCondition implements SDElement =
5    key(“assert”) Expression “;”;
6
7  SDVariableDeclaration implements
8    SDElement =
9    “let” MCType Name “=“
10   assignment:Expression “;”;
```



SD

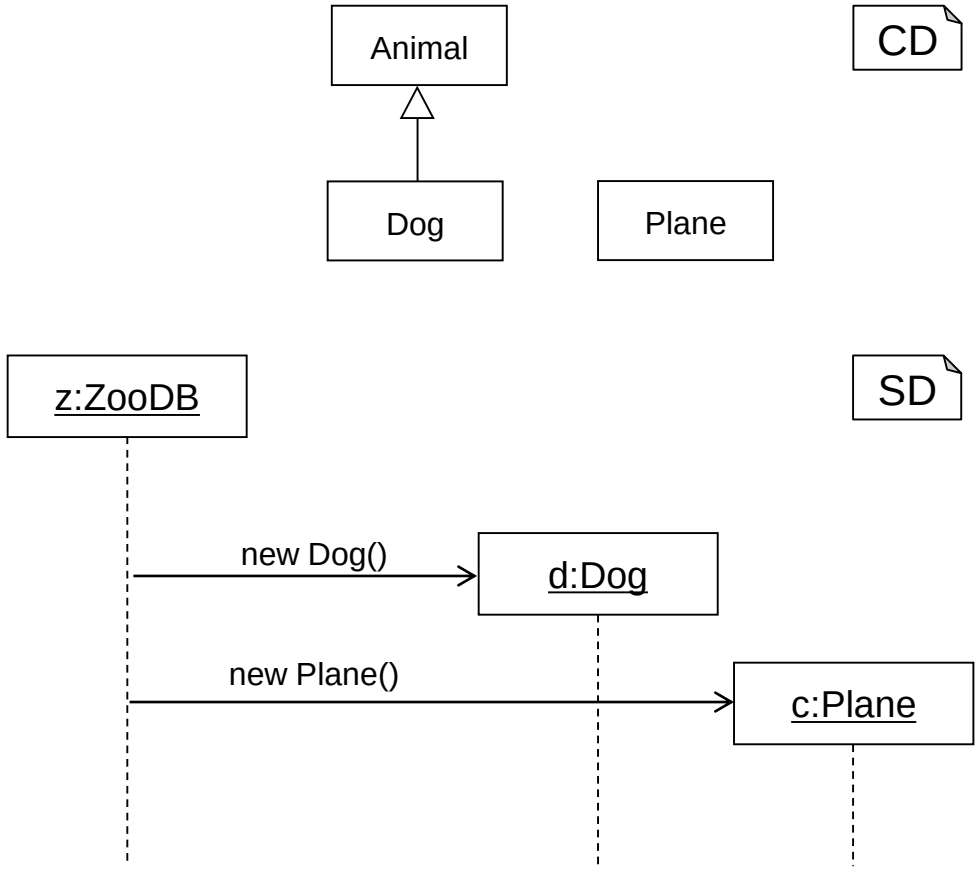
```
1  // ...
2
3  a -> b : foo(...);
4
5  let m = a.bar;
6
7  assert m == 2;
8
9  // ...
```

Context Conditions

Different Kinds of Context Conditions

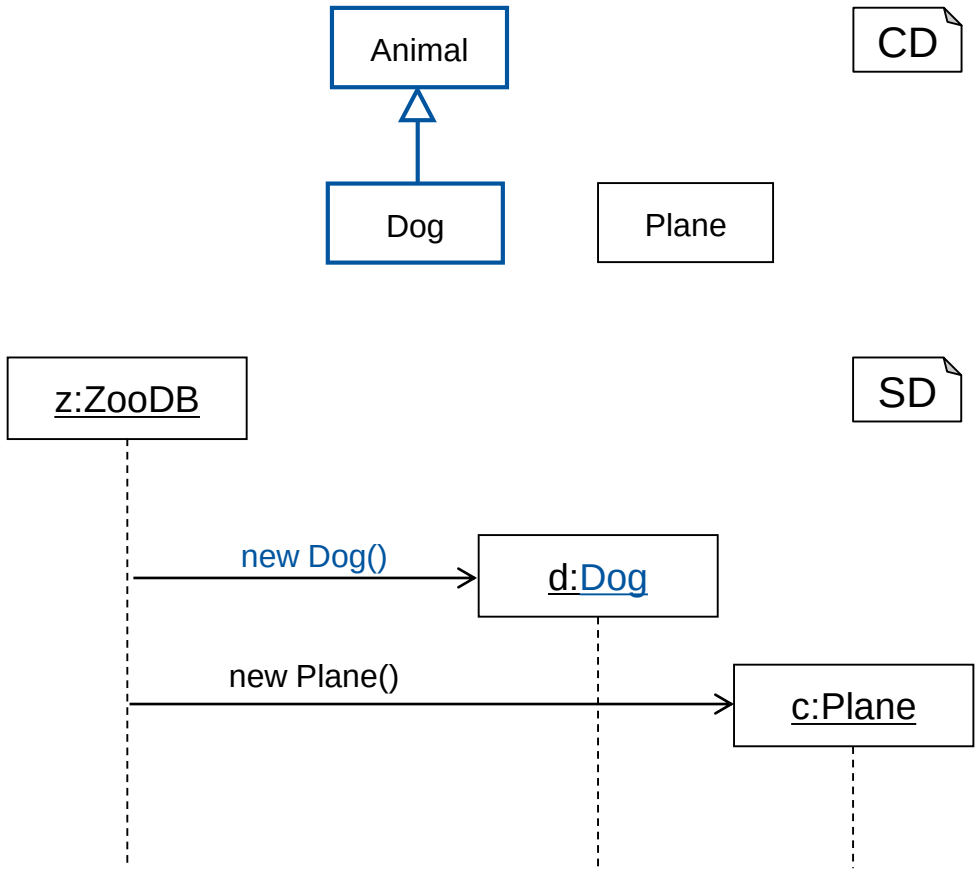
- General
  - Package name matches folder name
  - File extension is `.sd`
- Modifier
  - SequenceDiagram is complete, but any object is visible
- Interactions
  - Referenced implies Declared
  - Return call matches with a previous send call
  - Interaction has at least on source or target
  - Static methods are not invoked on objects
  - Method invoked exists in target type
- Objects
  - every object has a unique name
- Naming Conventions
  - Objects start with lower case
  - Types start with upper case
- Object Construction
  - Type checks

Type Check of Object Construction (invalid example)




```
1  sequencediagram MySD {                    SD
2    z : ZooDB;
3
4    z -> Animal d = new Dog();
5
6    z -> Animal c = new Plane();
7  }
```

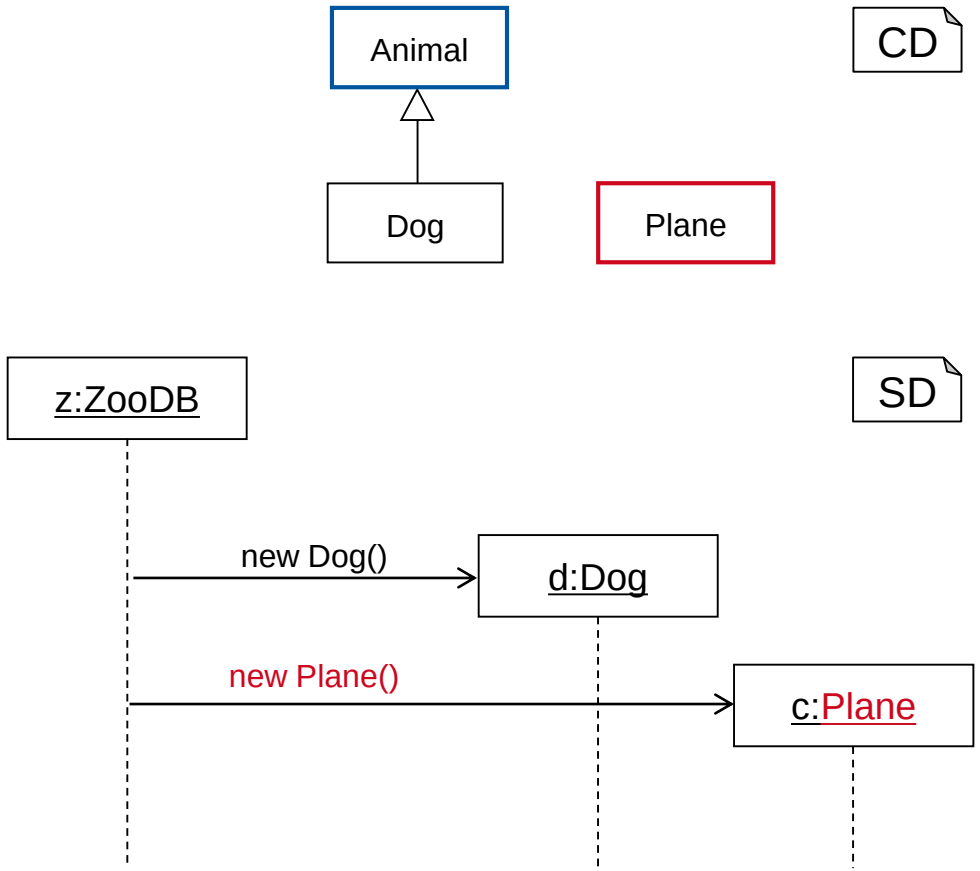
Type Check of Object Construction (invalid example)



SD

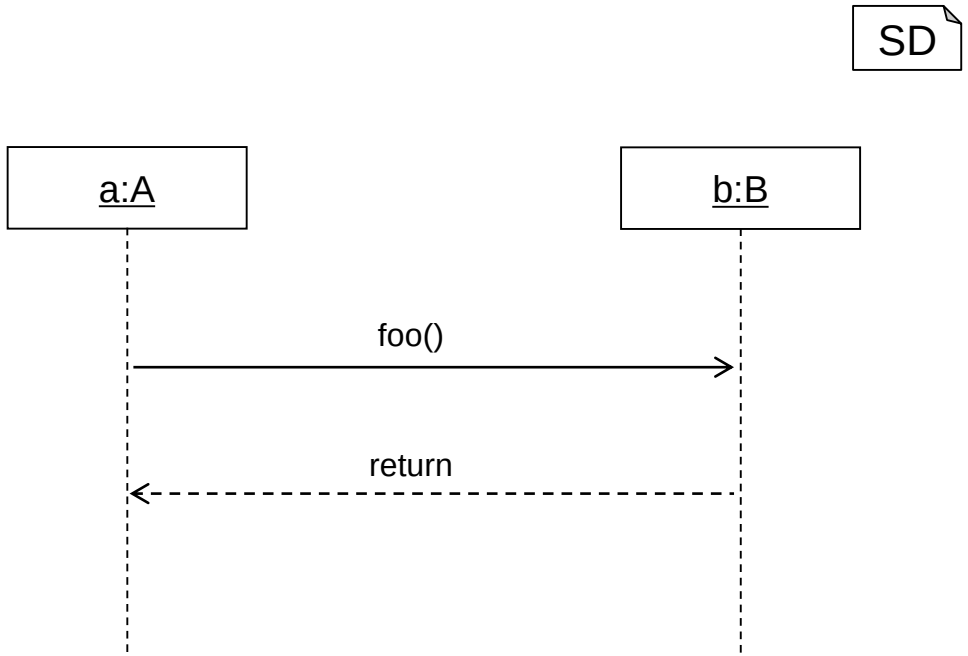
```
1  sequencediagram MySD {
2    z : ZooDB;
3
4    z -> Animal d = new Dog();
5
6    z -> Animal c = new Plane();
7  }
```

Type Check of Object Construction (invalid example)




```
1  sequencediagram MySD {                    SD
2    z : ZooDB;
3
4    z -> Animal d = new Dog();
5
6    z -> Animal c = new Plane();
7  }
```

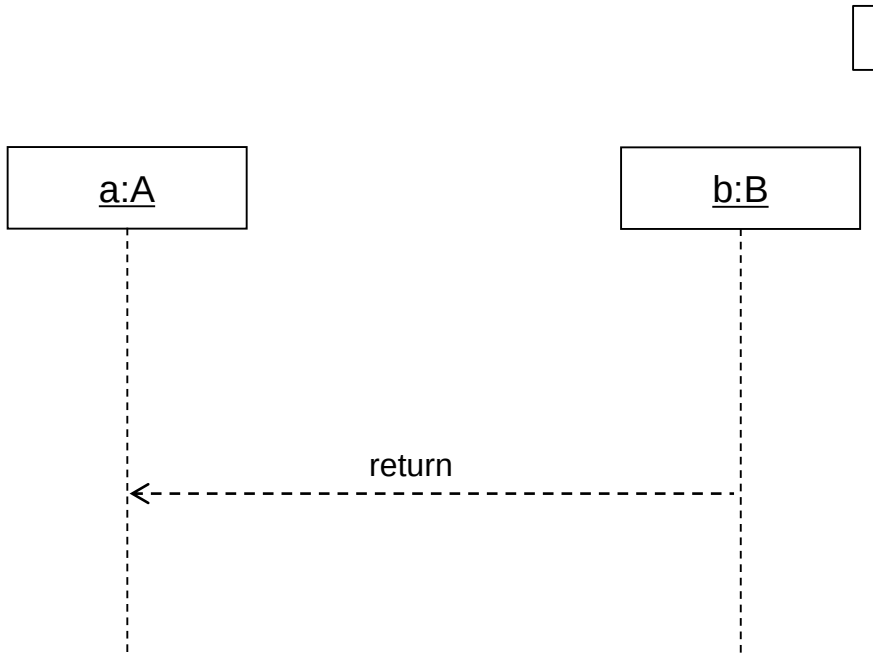
Return only after method call (valid example)




```
1  sequencediagram MySD {
2    a : A;
3    b : B;
4
5    a -> b : foo();
6    a <- b : return;
7  }
```

SD

Return only after method call (invalid example)

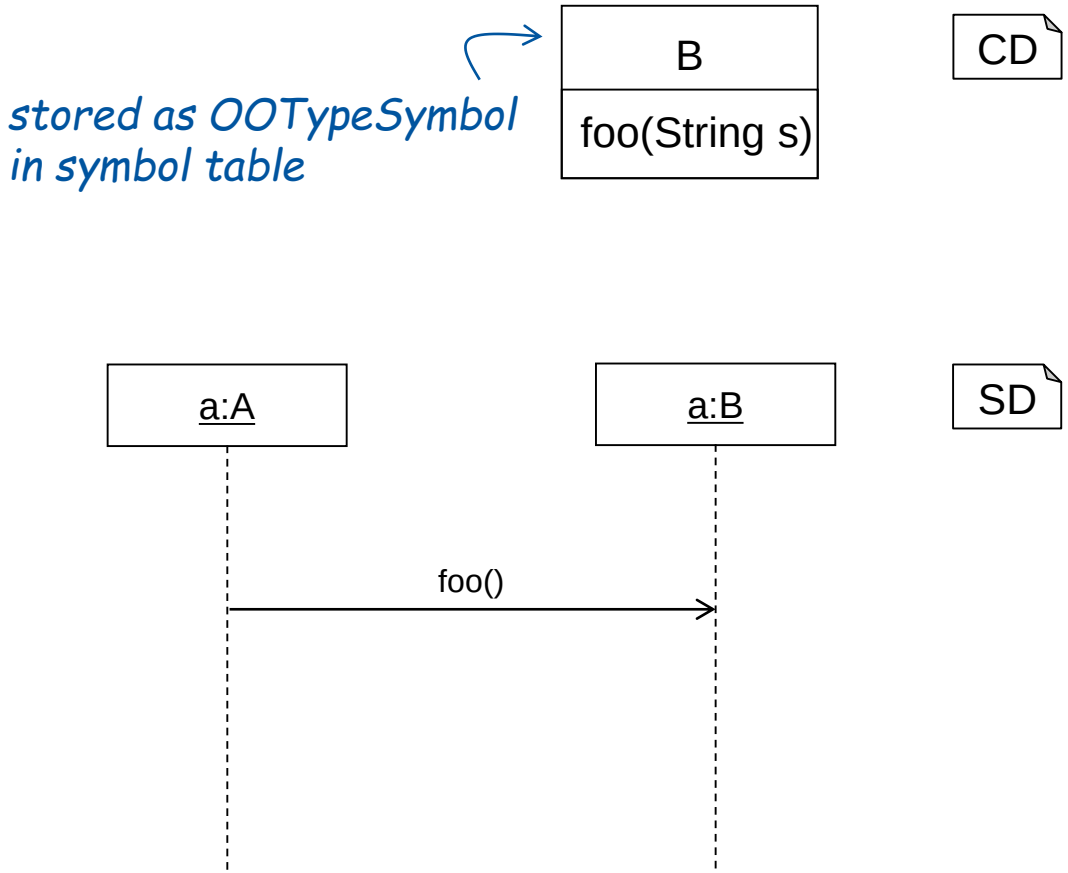



```
1  sequencediagram MySD {
2    a : A;
3    b : B;
4
5    // a -> b : foo();
6    a <- b : return;
7  }
```

SD

Christian Broering, Brian Sinkovec, Christian Volkmann

Method Action Valid (valid example)

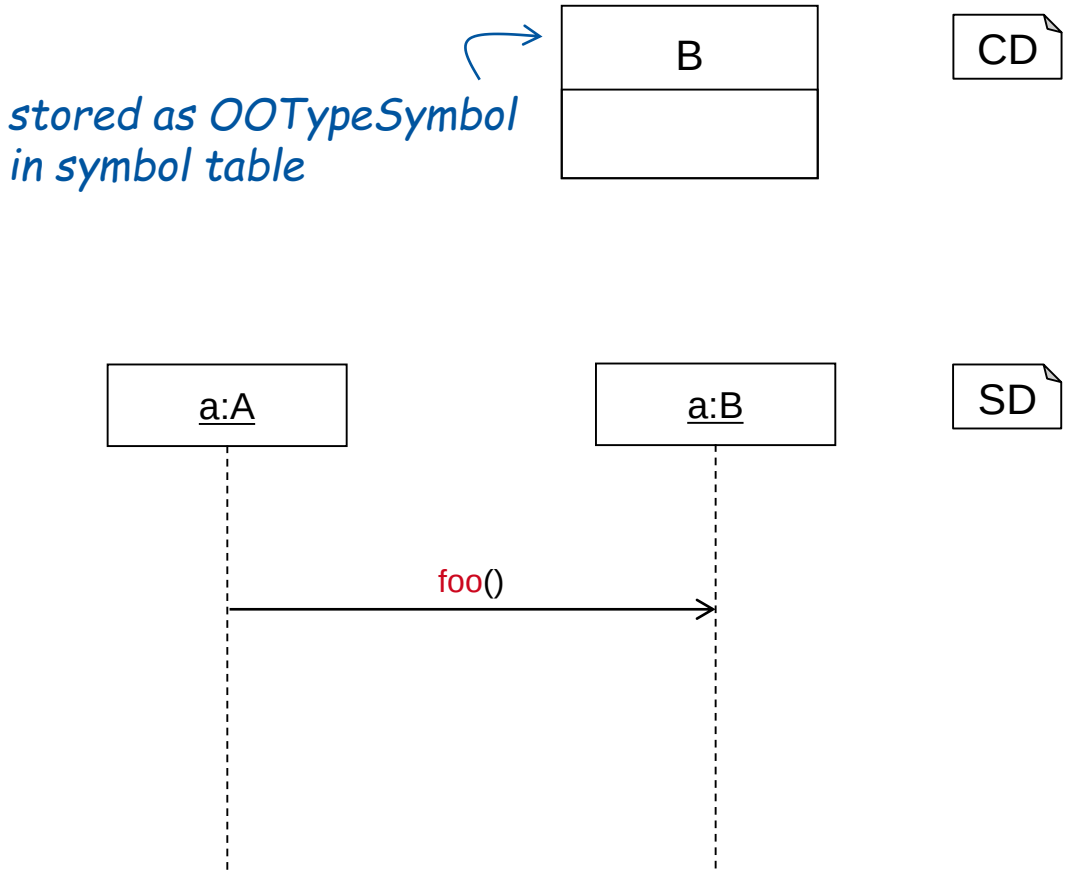



```
1  sequencediagram MySD {
2    a : A;
3    b : B;
4
5    a -> b : foo();
6
7  }
```

SD

Christian Broering, Brian Sinkovec, Christian Volkmann

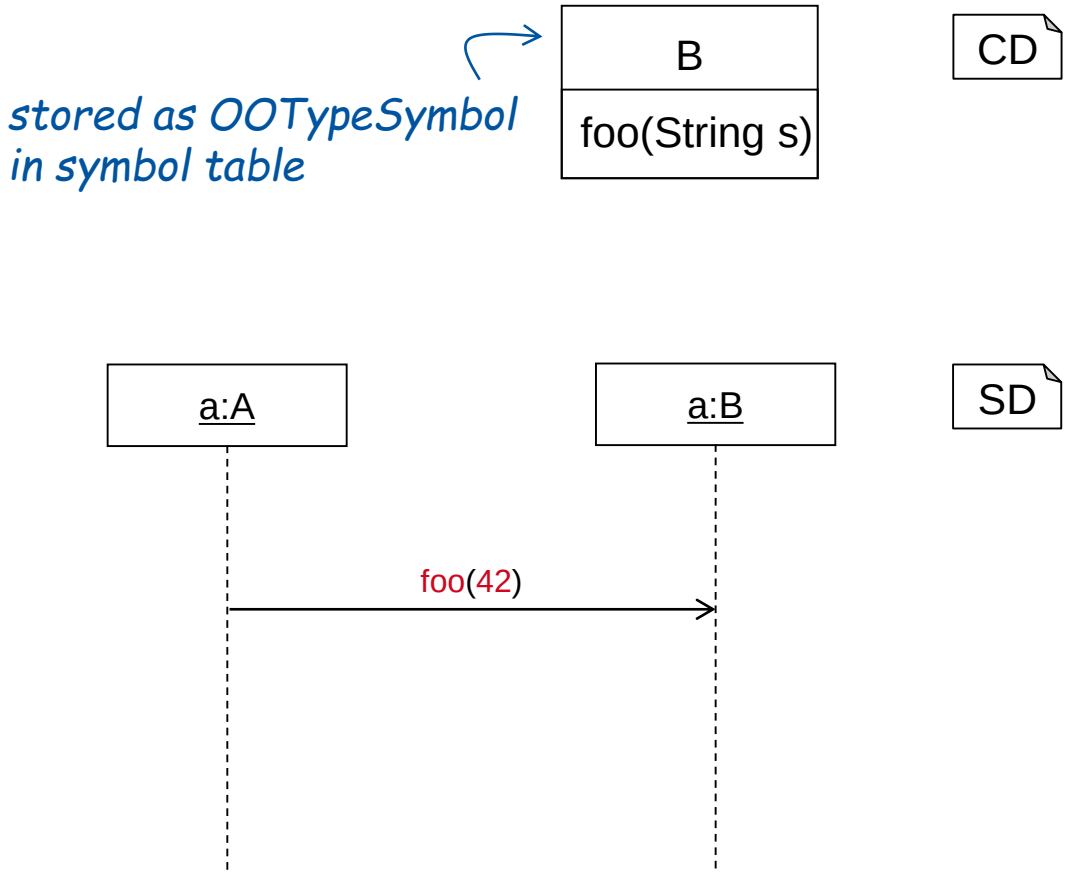
Method Action Valid (invalid example)




```
1  sequencediagram MySD {
2    a : A;
3    b : B;
4
5    a -> b : foo();
6
7  }
```

SD

Method Action Valid (invalid example)




```
1  sequencediagram MySD {
2    a : A;
3    b : B;
4
5    a -> b : foo(42);
6
7  }
```

SD

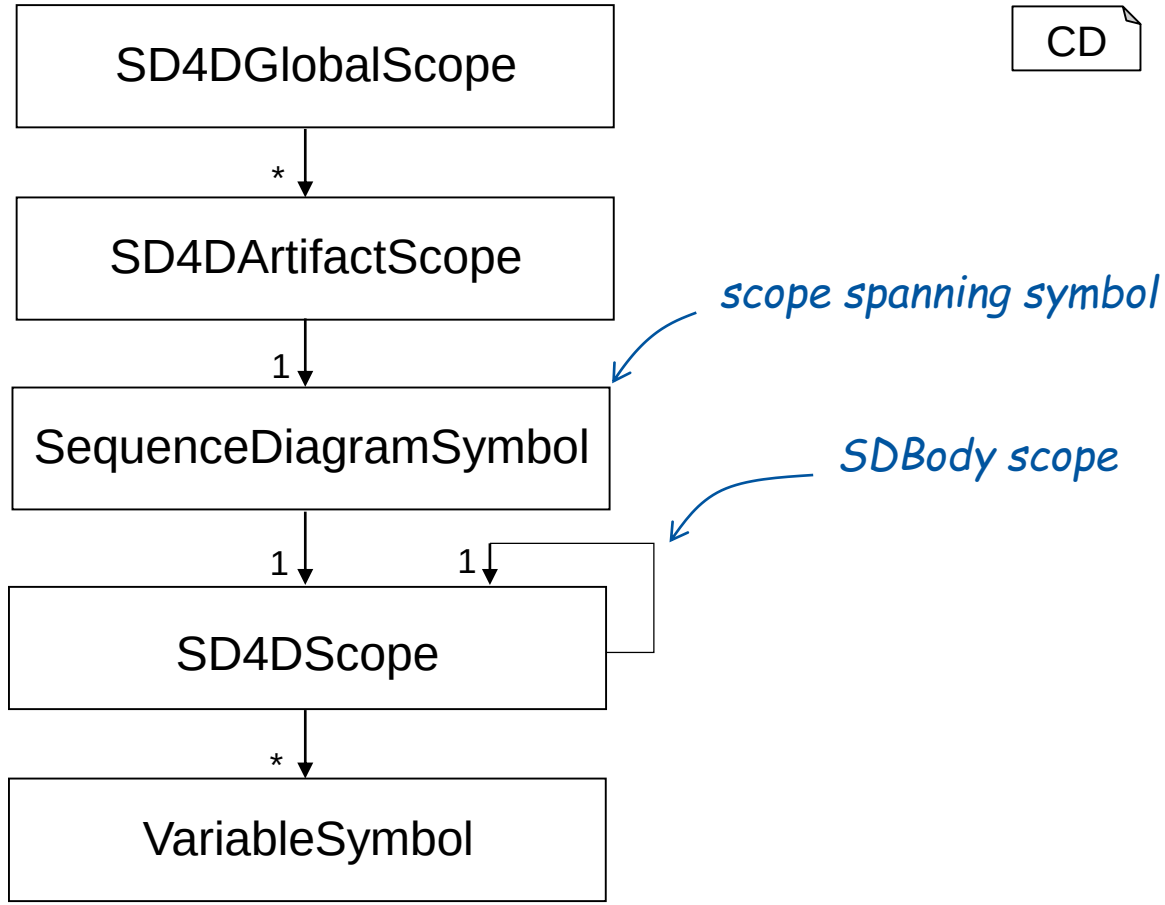
int, not a string

Christian Broering, Brian Sinkovec, Christian Volkmann

Symboltable

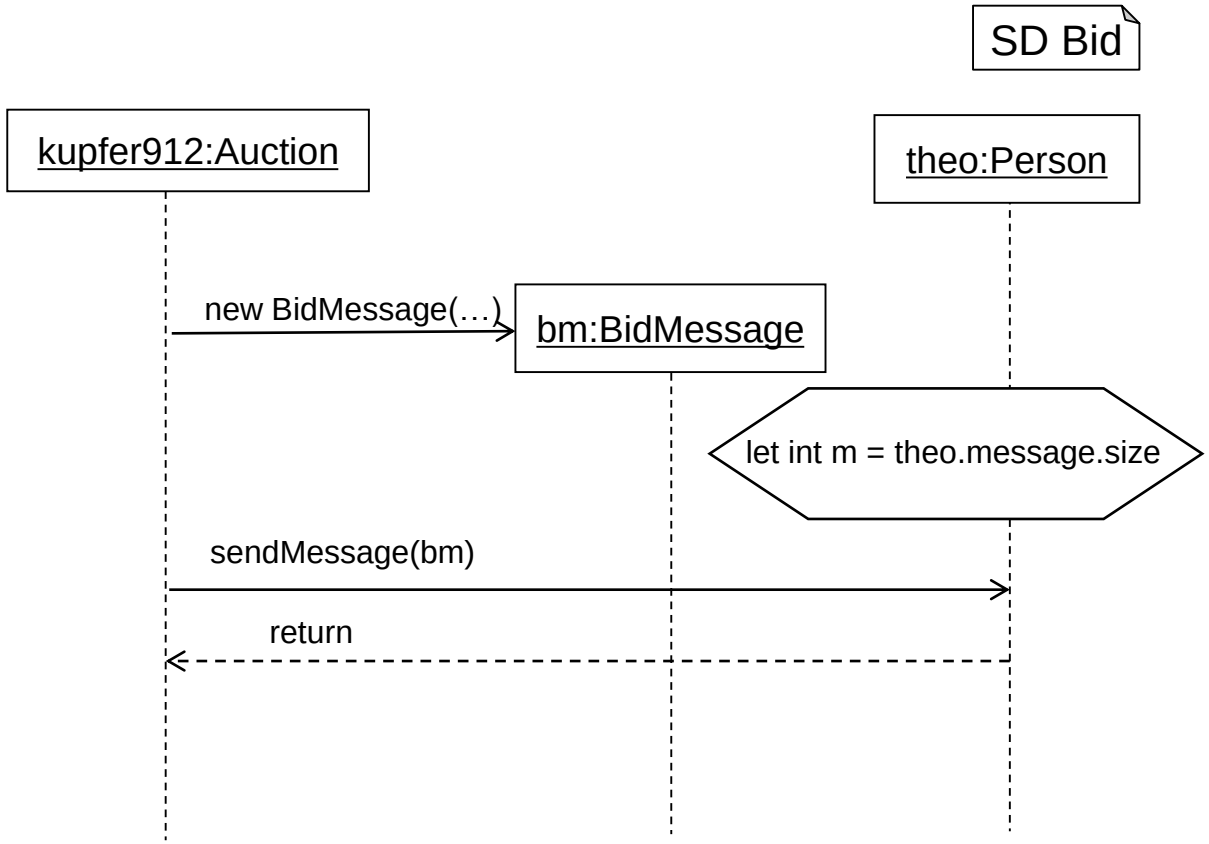
Symboltable Data Structure

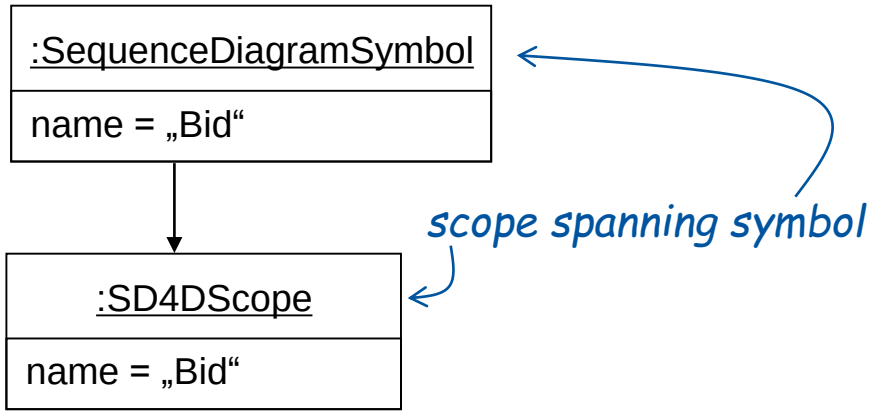
- Two symbol kinds
  - SequenceDiagramSymbol
  - VariableSymbol
- SequenceDiagramSymbol spans a scope
  - this scope spans another scope (SDBodyScope)



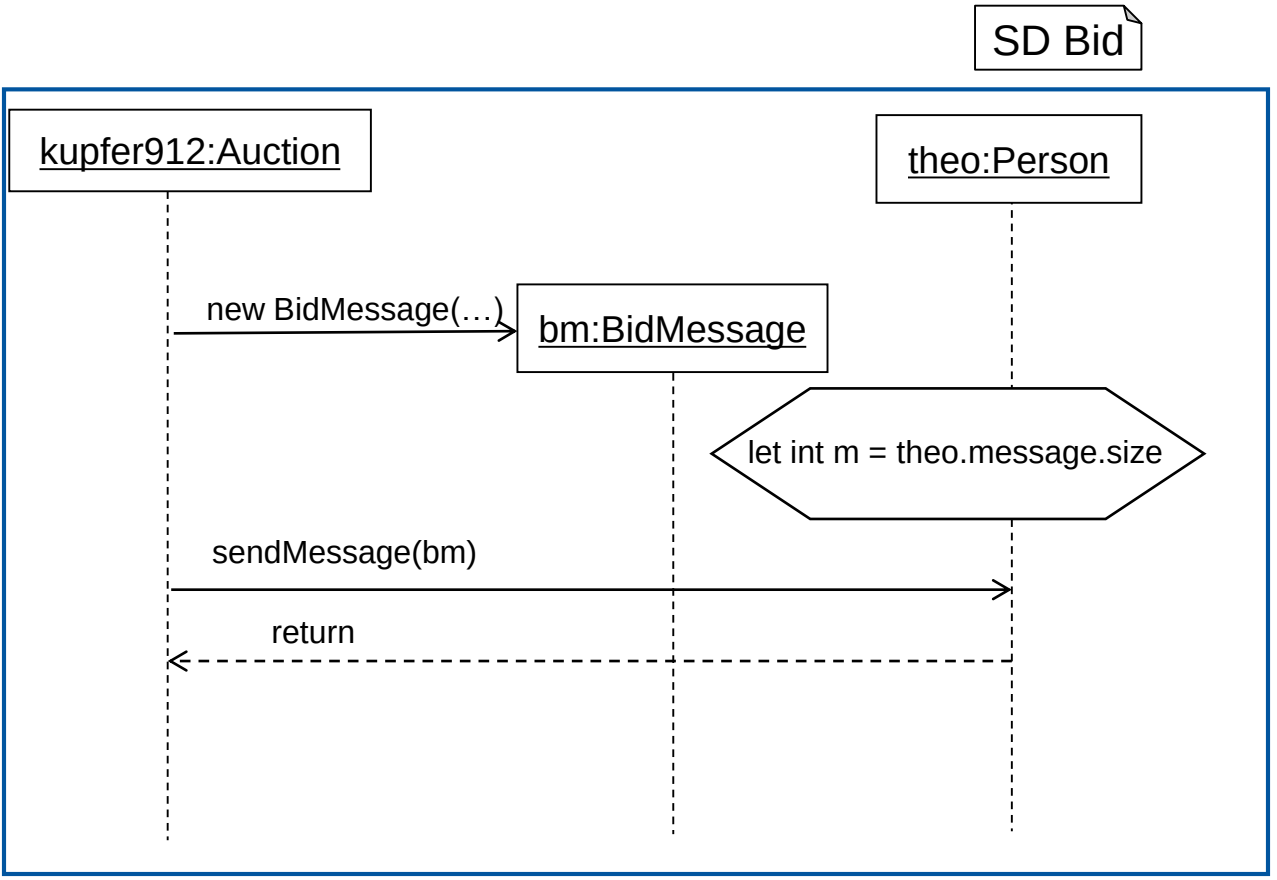
* SD4D short for SD4Development

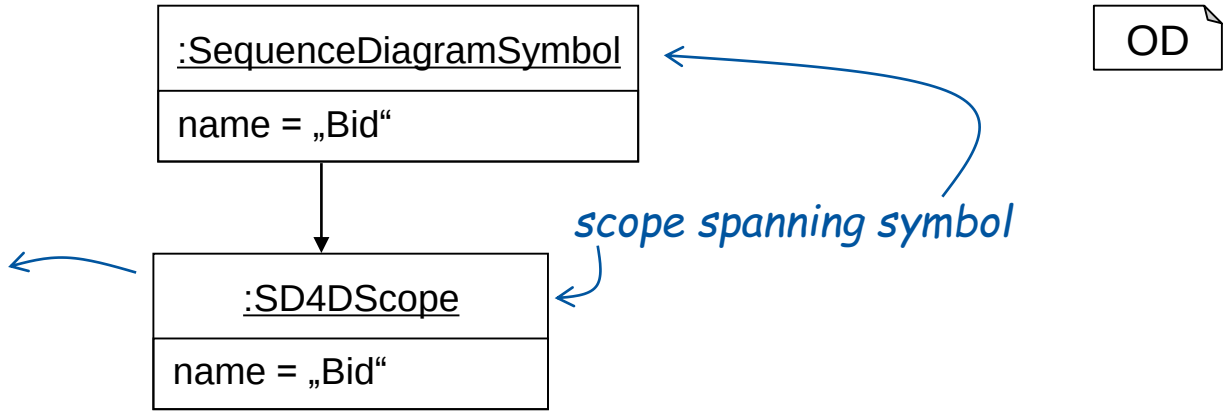
Example – Symboltable



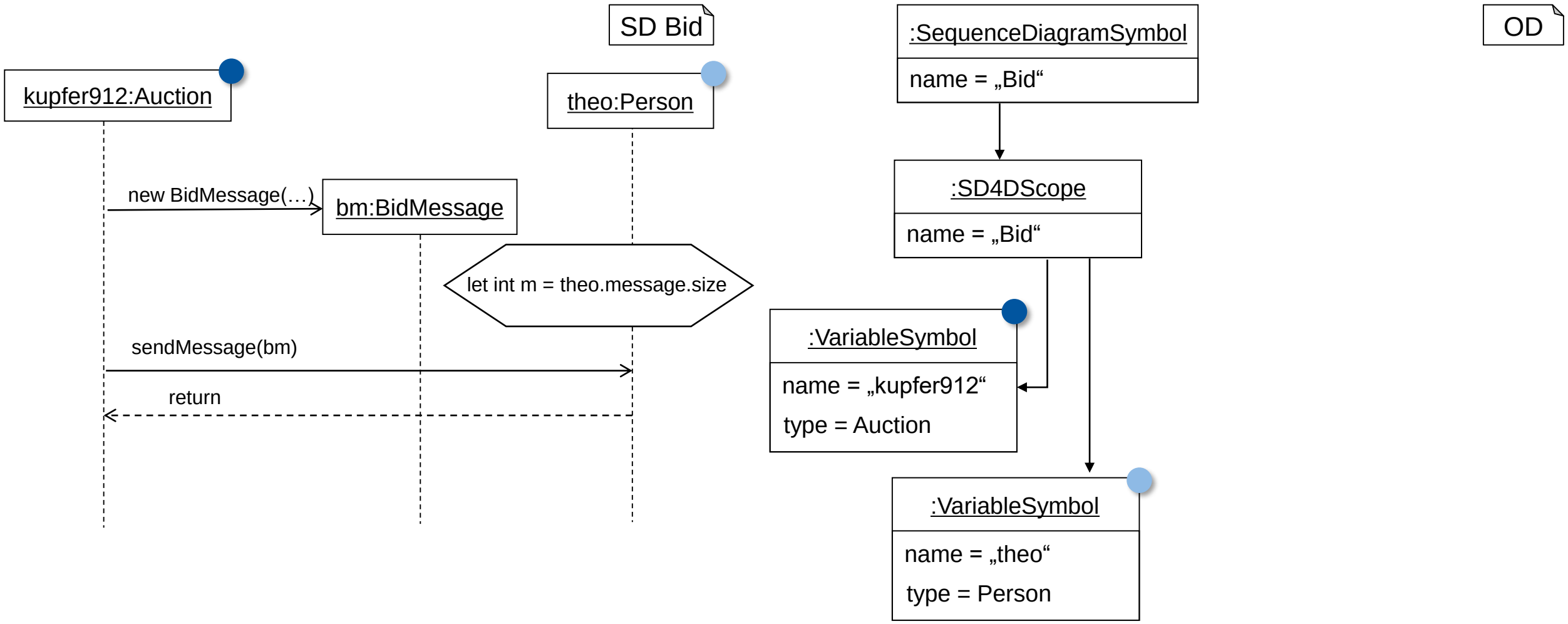


Example – Symboltable

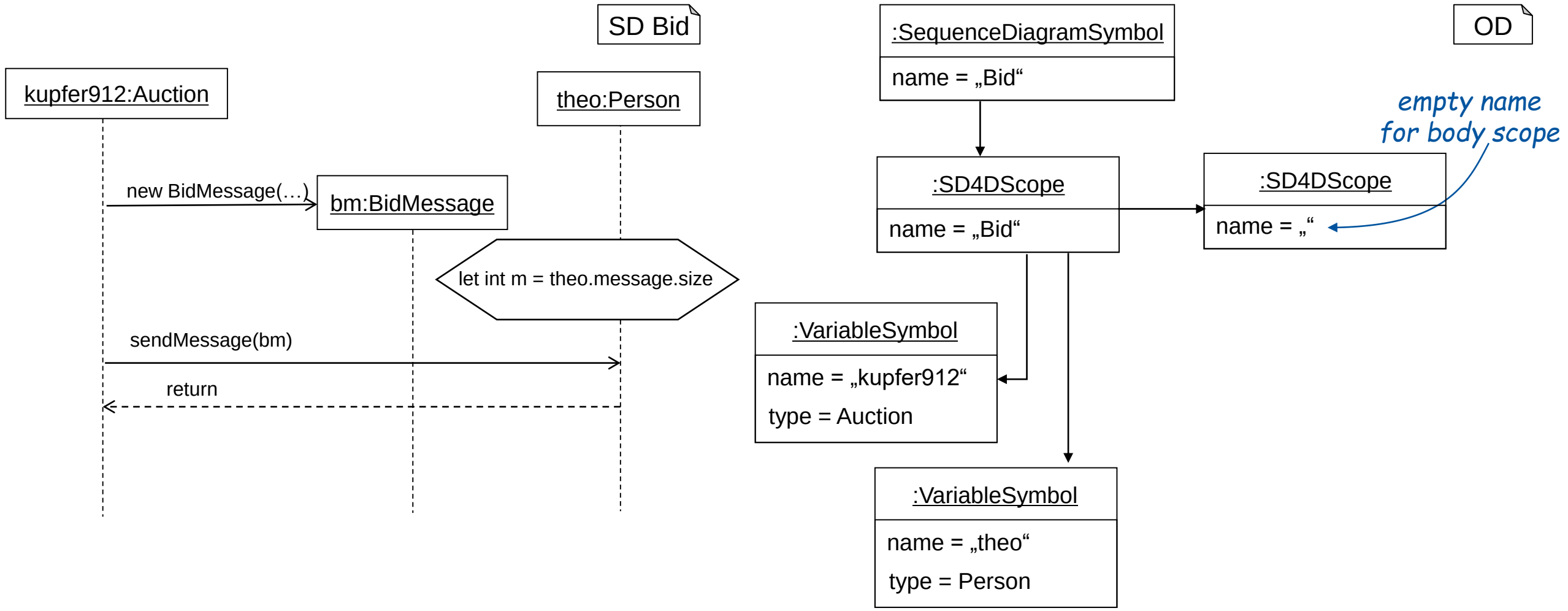




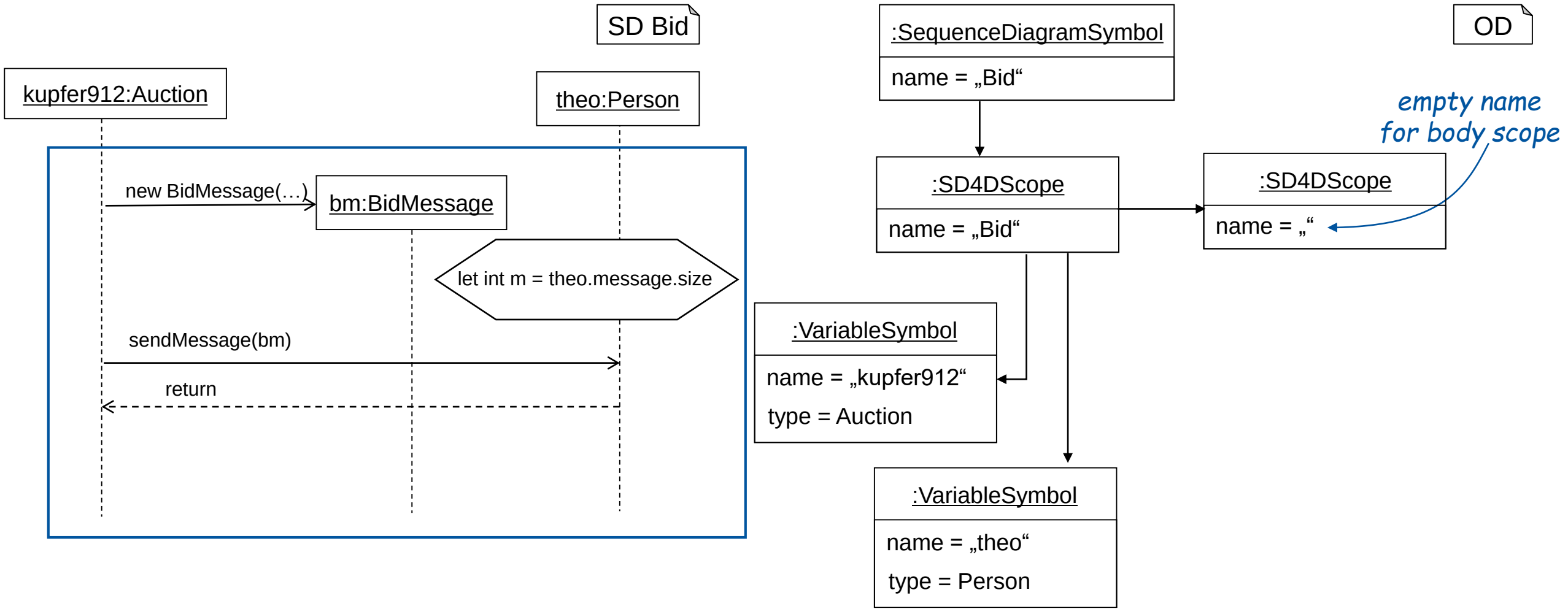
Example – Symboltable



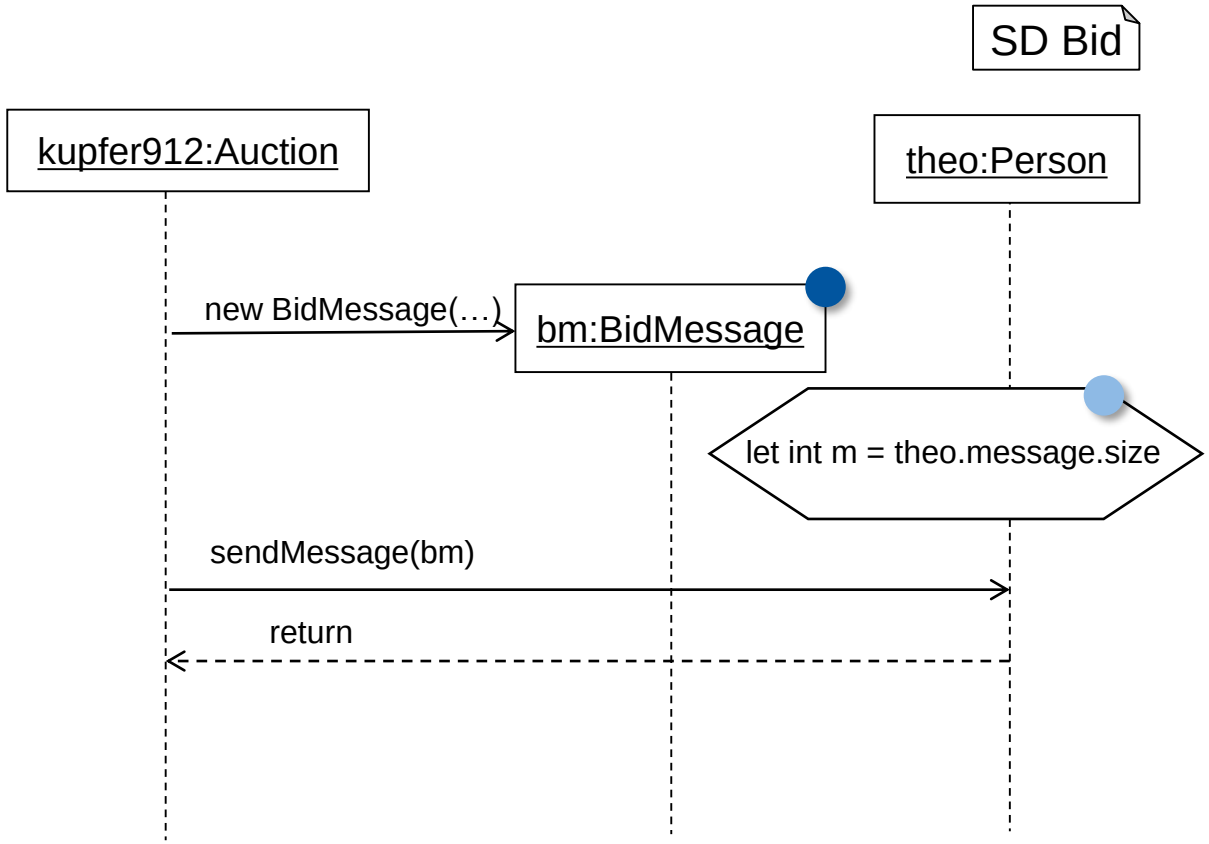
Example – Symboltable

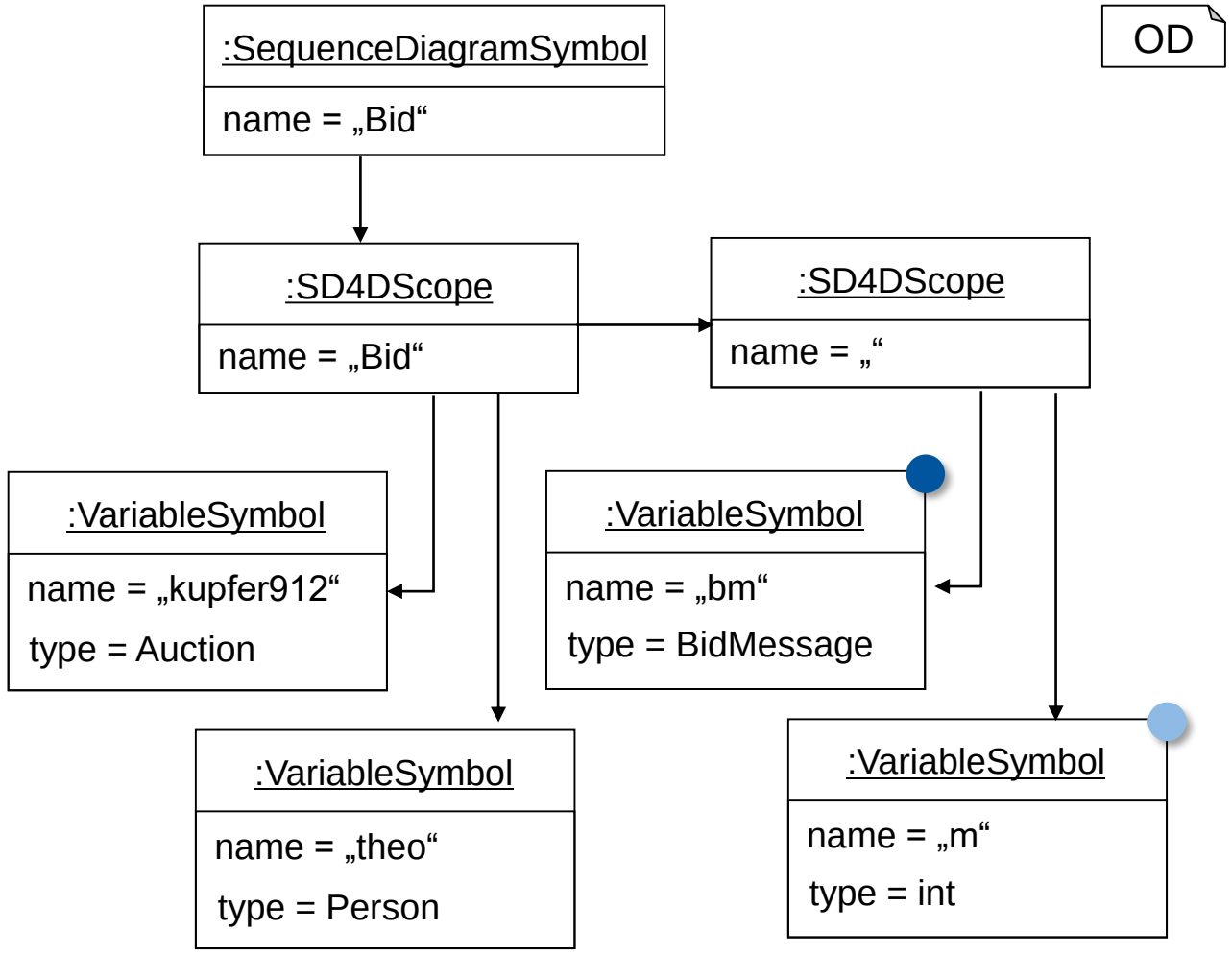


Example – Symboltable



Example – Symboltable





DeSer

DeSer in Action

SD

```
1  sequencediagram MySD {
2
3    kupfer912 : Auction;
4    // ...
5
6    kupfer912 -> BidMessage bm =
7                    new BidMessage();
8
9    let int m = bm.size;
10   // ...
11 }
```

deserialized

JSON

```
1  {
2    “kind”: “SD4DArtifactScope”,
3    “name”: “MySD”,
4    “sequenceDiagramSymbols”: [
5      { “kind”: “SequenceDiagramSymbol”, “name”: “MySD” }
6    ],
7    “subScopes”: [{
8      “kind”: “SD4DScope”,
9      “spanningSymbol”: { “kind”: “SequenceDiagramSymbol”, “name”: “MySD” },
10     “variableSymbols”: [
11       { “kind”: “VariableSymbol”, “name”: “kupfer912”, “type”: “Auction” }
12     ],
13     “subScopes”: [{
14       “kind”: “SD4DScope”,
15       “name”: “”,
16       “variableSymbols”: [
17         { “kind”: “VariableSymbol”, “name”: “bm”, “type”: “BidMessage” },
18         { “kind”: “VariableSymbol”, “name”: “m”,  “type”: “int” }
19       ]
20     }]
21   }]
22 }
```

serialized

DeSer in Action

SD

```
1  sequencediagram MySD {
2
3    kupfer912 : Auction;
4    // ...
5
6    kupfer912 -> BidMessage bm =
7                    new BidMessage();
8
9    let int m = bm.size;
10   // ...
11 }
```

JSON

```
1  {
2    “kind”: “SD4DArtifactScope”,
3    “name”: “MySD”,
4    “sequenceDiagramSymbols”: [
5      { “kind”: “SequenceDiagramSymbol”, “name”: “MySD” }
6    ],
7    “subScopes”: [{
8      “kind”: “SD4DScope”,
9      “spanningSymbol”: { “kind”: “SequenceDiagramSymbol”, “name”: “MySD” },
10     “variableSymbols”: [
11       { “kind”: “VariableSymbol”, “name”: “kupfer912”, “type”: “Auction” }
12     ],
13     “subScopes”: [{
14       “kind”: “SD4DScope”,
15       “name”: “”,
16       “variableSymbols”: [
17         { “kind”: “VariableSymbol”, “name”: “bm”, “type”: “BidMessage” },
18         { “kind”: “VariableSymbol”, “name”: “m”,  “type”: “int” }
19       ]
20     }]
21   }]
22 }
```

DeSer in Action

SD

```
1  sequencediagram MySD {
2
3    kupfer912 : Auction;
4    // ...
5
6    kupfer912 -> BidMessage bm =
7                    new BidMessage();
8
9    let int m = bm.size;
10   // ...
11 }
```

JSON

```
1  {
2    “kind”: “SD4DArtifactScope”,
3    “name”: “MySD”,
4    “sequenceDiagramSymbols”: [
5      { “kind”: “SequenceDiagramSymbol”, “name”: “MySD” }
6    ],
7    “subScopes”: [{
8      “kind”: “SD4DScope”,
9      “spanningSymbol”: { “kind”: “SequenceDiagramSymbol”, “name”: “MySD” },
10     “variableSymbols”: [
11       { “kind”: “VariableSymbol”, “name”: “kupfer912”, “type”: “Auction” }
12     ],
13     “subScopes”: [{
14       “kind”: “SD4DScope”,
15       “name”: “”,
16       “variableSymbols”: [
17         { “kind”: “VariableSymbol”, “name”: “bm”, “type”: “BidMessage” },
18         { “kind”: “VariableSymbol”, “name”: “m”,  “type”: “int” }
19       ]
20     }]
21   }]
22 }
```

DeSer in Action

SD

```
1  sequencediagram MySD {
2
3    kupfer912 : Auction;
4    // ...
5
6    kupfer912 -> BidMessage bm =
7                    new BidMessage();
8
9    let int m = bm.size;
10   // ...
11 }
```

JSON

```
1  {
2    “kind”: “SD4DArtifactScope”,
3    “name”: “MySD”,
4    “sequenceDiagramSymbols”: [
5      { “kind”: “SequenceDiagramSymbol”, “name”: “MySD” }
6    ],
7    “subScopes”: [{
8      “kind”: “SD4DScope”,
9      “spanningSymbol”: { “kind”: “SequenceDiagramSymbol”, “name”: “MySD” },
10     “variableSymbols”: [
11       { “kind”: “VariableSymbol”, “name”: “kupfer912”, “type”: “Auction” }
12     ],
13     “subScopes”: [{
14       “kind”: “SD4DScope”,
15       “name”: “”,
16       “variableSymbols”: [
17         { “kind”: “VariableSymbol”, “name”: “bm”, “type”: “BidMessage” },
18         { “kind”: “VariableSymbol”, “name”: “m”,  “type”: “int” }
19       ]
20     }]
21   }]
22 }
```

DeSer in Action

SD

```
1  sequencediagram MySD {
2
3    kupfer912 : Auction;
4    // ...
5
6    kupfer912 -> BidMessage bm =
7                    new BidMessage();
8
9    let int m = bm.size;
10   // ...
11 }
```

JSON

```
1  {
2    “kind”: “SD4DArtifactScope”,
3    “name”: “MySD”,
4    “sequenceDiagramSymbols”: [
5      { “kind”: “SequenceDiagramSymbol”, “name”: “MySD” }
6    ],
7    “subScopes”: [{
8      “kind”: “SD4DScope”,
9      “spanningSymbol”: { “kind”: “SequenceDiagramSymbol”, “name”: “MySD” },
10     “variableSymbols”: [
11       { “kind”: “VariableSymbol”, “name”: “kupfer912”, “type”: “Auction” }
12     ],
13     “subScopes”: [{
14       “kind”: “SD4DScope”,
15       “name”: “”,
16       “variableSymbols”: [
17         { “kind”: “VariableSymbol”, “name”: “bm”, “type”: “BidMessage” },
18         { “kind”: “VariableSymbol”, “name”: “m”,  “type”: “int” }
19       ]
20     }]
21   }]
22 }
```

Summary

SDInteraction, SDSendMessage

MC

```
interface SDInteraction
  extends SDElement;

SDSendMessage
  implements SDInteraction  =
    SDSource? "->" SDTarget? ":"
    SDAction (";" | SDActivityBar)
;

interface SDSource;
interface SDTarget;

interface SDAction;

SDActivityBar = "{" SDElement* "}";
```

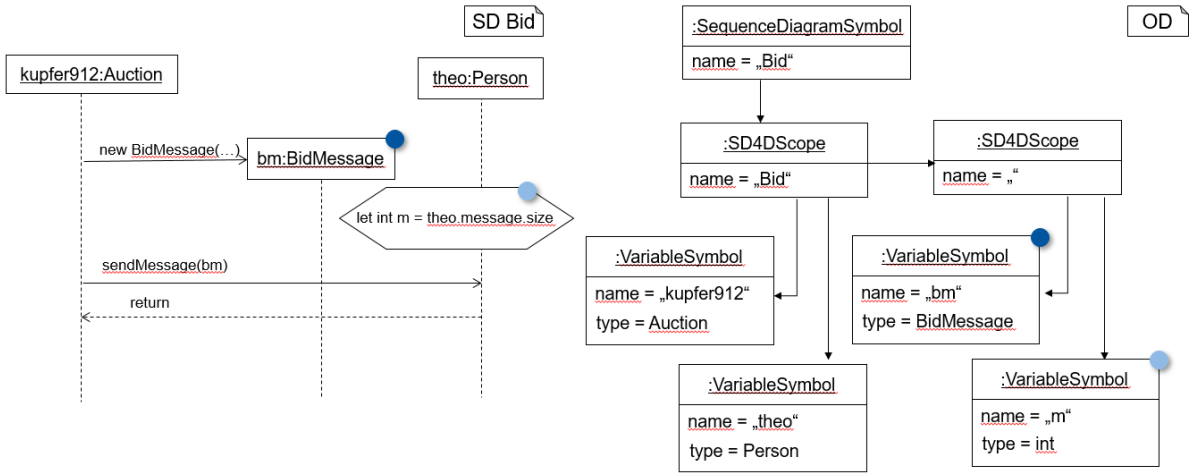
SD

```
// ...

a -> b : foo();

a -> b : bar() {
  b -> c : foobar();
}

// ...
```

Different Kinds of Context Conditions

- General
  - Package name matches folder name
  - File extension is .sd
- Modifier
  - SequenceDiagram is complete, but any object is visible
- Interactions
  - Referenced implies Declared
  - Return call matches with a previous send call
  - Interaction has at least on source or target
  - Static methods are not invoked on objects
  - Method invoked exists in target type
- Objects
  - every object has a unique name
- Naming Conventions
  - Objects start with lower case
  - Types start with upper case
- Object Construction
  - Type checks

Example – Symboltable



DeSer in Action

SD

```
sequencediagram MySD {

  kupfer912 : Auction;
  // ...

  kupfer912 -> BidMessage bm =
                    new BidMessage();

  let int m = bm.size;
  // ...
}
```

⟺

JSON

```
{
  "kind": "SD4DArtifactScope",
  "name": "MySD",
  "sequenceDiagramSymbols": [
    { "kind": "SequenceDiagramSymbol", "name": "MySD" }
  ],
  "subScopes": [{
    "kind": "SD4DScope",
    "spanningSymbol": { "kind": "SequenceDiagramSymbol", "name": "MySD" },
    "variableSymbols": [
      { "kind": "VariableSymbol", "name": "kupfer912", "type": "Auction" }
    ],
    "subScopes": [{
      "kind": "SD4DScope",
      "name": "",
      "variableSymbols": [
        { "kind": "VariableSymbol", "name": "bm", "type": "BidMessage" },
        { "kind": "VariableSymbol", "name": "m",  "type": "int" }
      ]
    }]
  }]
}
```