# Comparer Examples

Class `LionWeb.Utils.Comparer` deeply compares two lists of nodes with detailed report on differences. It considers all input nodes, including all descendants and annotations.

This document lists examples for each compared element.

We assume familiarity with LionWeb Meta-metamodel (M3).

# M2 (Metamodel)

## Basic elements

Node *ids*, metamodel element *keys*, and language *versions* are strings. We consider *ids*, *keys*, and *versions* equal if their string values are equal.

## Language

We consider two languages equal if both their key and their version match.

**NOTE** Each box represents *one language*.

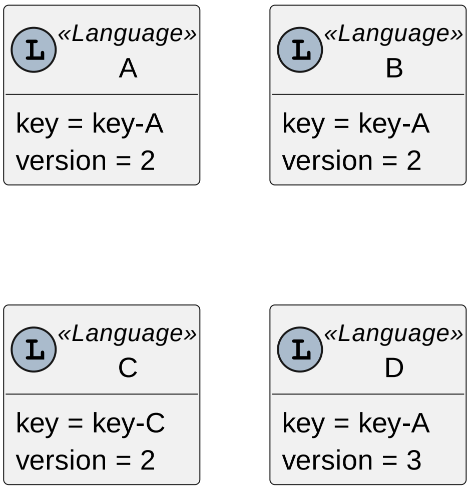

*Figure 1. Language examples*

- `A` and `B` are equal because both their *key* and *version* matches.
- `A` and `C` are not equal because they have different *keys*: `key-A` vs. `key-C`.
- `A` and `D` are not equal because they have different *versions*: `2` vs. `3`.

## Classifier

We consider classifiers (i.e. concepts and annotations) equal if their variant, key, and language match. *variant* describes whether the classifier is an instance of Annotation or Concept.

**NOTE** Each box represents *one language* or *one classifier*.

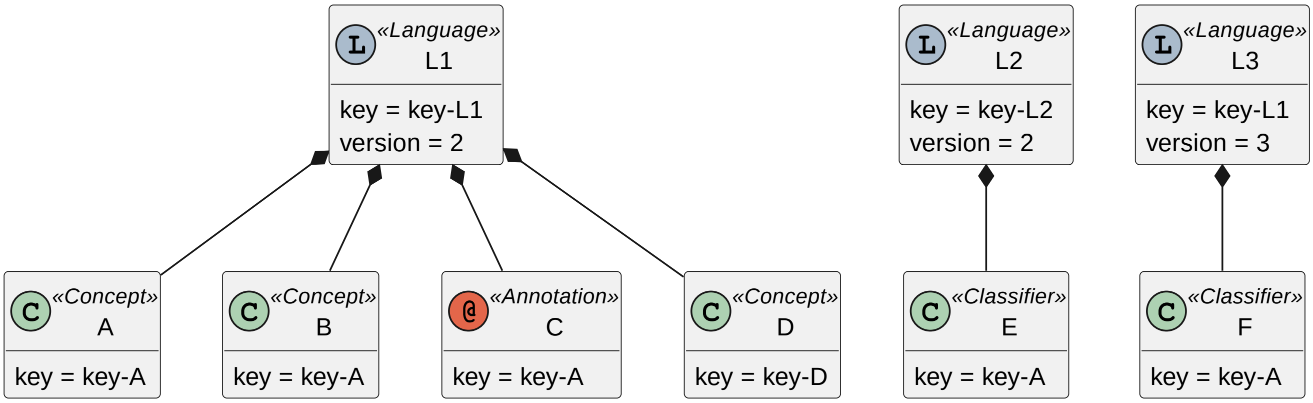

*Figure 2. Classifier examples*

- `A` and `B` are equal because they are both *Concepts*, have equal *keys*, and belong to equal *languages*.
- `A` and `C` are not equal because `A` is a *Concept*, but `C` is an *Annotation*.
- `A` and `D` are not equal because they have different *keys*: `key-A` vs. `key-D`.
- `A` and `E` are not equal because they belong to different *languages* (in terms of *key*: `key-L1` vs. `key-L2`).
- `A` and `F` are not equal because they belong to different *languages* (in terms of *version*: `2` vs. `3`).

# Feature

We consider features (i.e. properties, containments, and references) equal if their variant, key, and classifier match. *variant* describes whether the feature is an instance of Property, Containment, or Reference.

**NOTE** Each box represents *one language*, *one classifier*, or *one feature*.

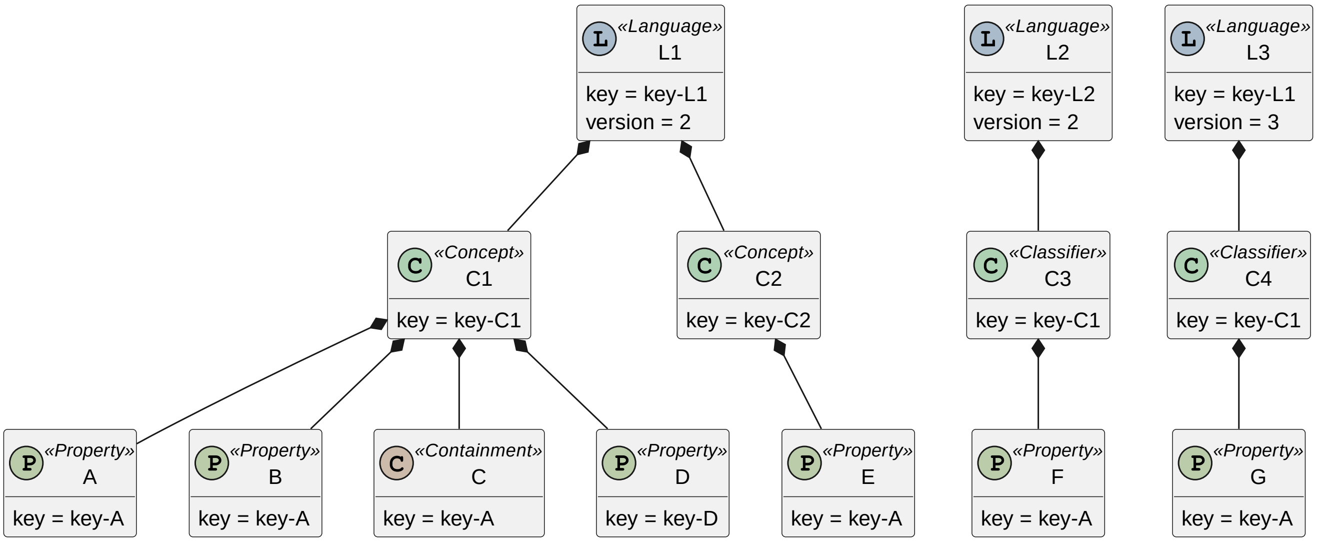

*Figure 3. Feature examples*

- `A` and `B` are equal because they are both *Properties*, have equal *keys*, and belong to equal *concepts*.
- `A` and `C` are not equal because `A` is a *Property* but `C` is a *Containment*.
- `A` and `D` are not equal because they have different *keys*: `key-A` vs. `key-D`.
- `A` and `E` are not equal because they belong to different *classifiers* (in terms of *key*: `key-C1` vs. `key-C2`).
- `A` and `F` are not equal because they belong to different *classifiers* (in terms of *language key*: `key-L1` vs. `key-L2`).
- `A` and `G` are not equal because they belong to different *classifiers* (in terms of *language version*: `2` vs. `3`).

# M1 (Model)

## Property

We consider properties equal if their property and value match.

The property value can be of type

**string**

Equal by string comparison.

**integer**

Equal by int comparison.

**boolean**

Equal by bool comparison.

**enumeration**

Equal if same C# enum type, and same literal name.

We compare enum type by C# `==` operator. We compare enumeration literal names by string comparison.

**NOTE** Each box represents *one property*.

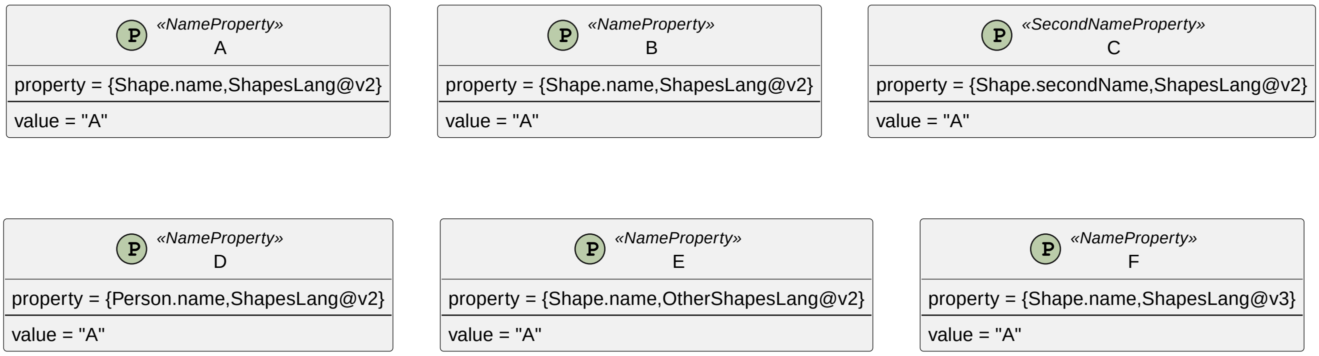
P
«NameProperty»
A
property = {Shape.name,ShapesLang@v2}
value = "A"
P
«NameProperty»
B
property = {Shape.name,ShapesLang@v2}
value = "A"
P
«SecondNameProperty»
C
property = {Shape.secondName,ShapesLang@v2}
value = "A"
P
«NameProperty»
D
property = {Person.name,ShapesLang@v2}
value = "A"
P
«NameProperty»
E
property = {Shape.name,OtherShapesLang@v2}
value = "A"
P
«NameProperty»
F
property = {Shape.name,ShapesLang@v3}
value = "A"

*Figure 4. Simple property examples*

- `A` and `B` are equal because they have equal *property*, and equal *values*.
- `A` and `C` are not equal because they have different *properties* (in terms of *property key*: `name` vs. `secondName`).
- `A` and `D` are not equal because they have different *properties* (in terms of *concept key*: `Shape` vs. `Person`).
- `A` and `E` are not equal because they have different *properties* (in terms of *language key*: `ShapesLang` vs. `OtherShapesLang`).
- `A` and `F` are not equal because they have different *properties* (in terms of *language version*: `2` vs. `3`).

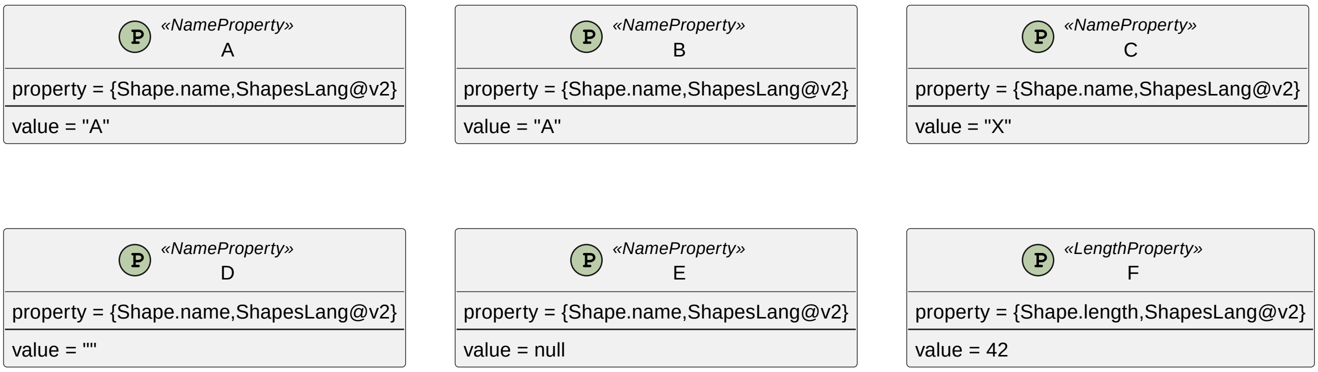
P
«NameProperty»
A
property = {Shape.name,ShapesLang@v2}
value = "A"
P
«NameProperty»
B
property = {Shape.name,ShapesLang@v2}
value = "A"
P
«NameProperty»
C
property = {Shape.name,ShapesLang@v2}
value = "X"
P
«NameProperty»
D
property = {Shape.name,ShapesLang@v2}
value = ""
P
«NameProperty»
E
property = {Shape.name,ShapesLang@v2}
value = null
P
«LengthProperty»
F
property = {Shape.length,ShapesLang@v2}
value = 42

*Figure 5. Property string value examples*

- `A` and `B` are equal because they have equal *values*.
- `A` and `C` are not equal because they have different *values*: `A` vs. `X`.
- `A` and `D` are not equal because they have different *values*: `A` vs. empty string.
- `A` and `E` are not equal because they have different *values*: `A` vs. null.
- `A` and `F` are not equal because they have different *values*: `A` vs. (integer) 42. They also have different *property keys*, as we cannot have an integer value in a string property.

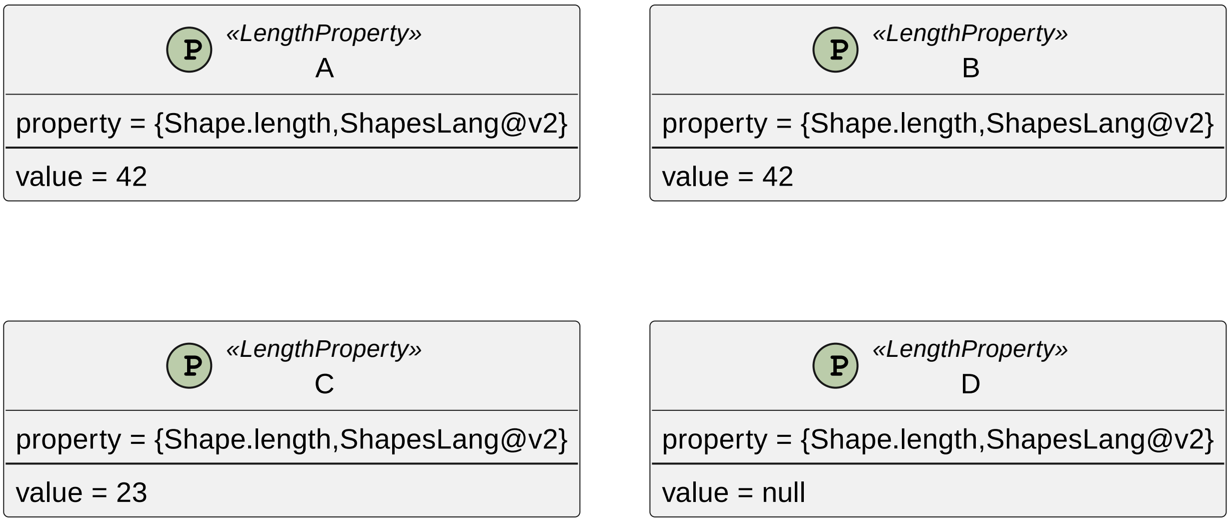
P
«LengthProperty»
A
property = {Shape.length,ShapesLang@v2}
value = 42
P
«LengthProperty»
B
property = {Shape.length,ShapesLang@v2}
value = 42
P
«LengthProperty»
C
property = {Shape.length,ShapesLang@v2}
value = 23
P
«LengthProperty»
D
property = {Shape.length,ShapesLang@v2}
value = null

*Figure 6. Property integer value examples*

- `A` and `B` are equal because they have equal *values*.
- `A` and `C` are not equal because they have different *values*: `42` vs. `23`.
- `A` and `D` are not equal because they have different *values*: `42` vs. null.

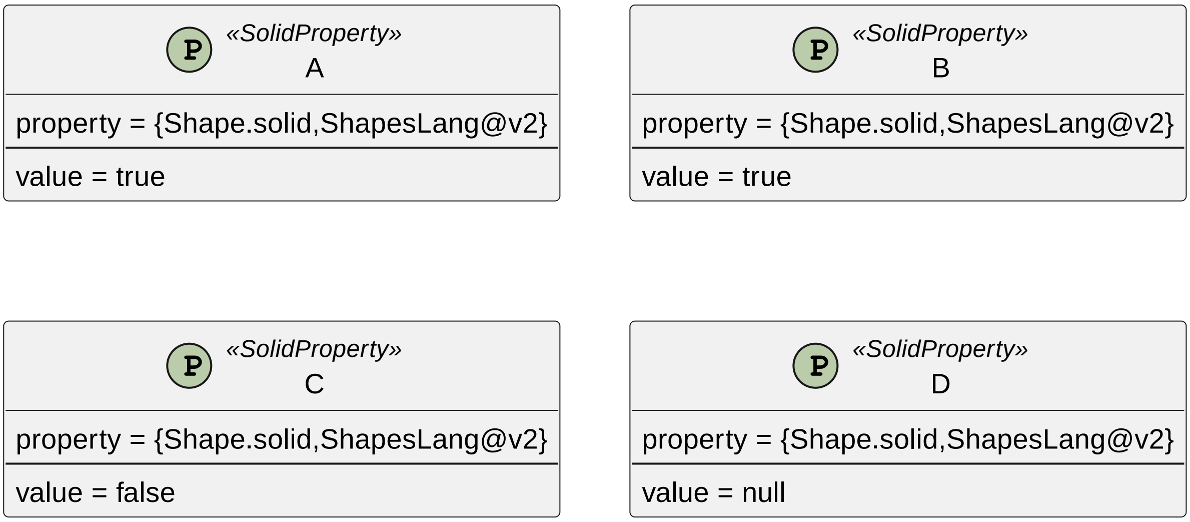
P
«SolidProperty»
A
property = {Shape.solid,ShapesLang@v2}
value = true
P
«SolidProperty»
B
property = {Shape.solid,ShapesLang@v2}
value = true
P
«SolidProperty»
C
property = {Shape.solid,ShapesLang@v2}
value = false
P
«SolidProperty»
D
property = {Shape.solid,ShapesLang@v2}
value = null

*Figure 7. Property boolean value examples*

- `A` and `B` are equal because they have equal *values*.
- `A` and `C` are not equal because they have different *values*: `true` vs. `false`.
- `A` and `D` are not equal because they have different *values*: `true` vs. null.

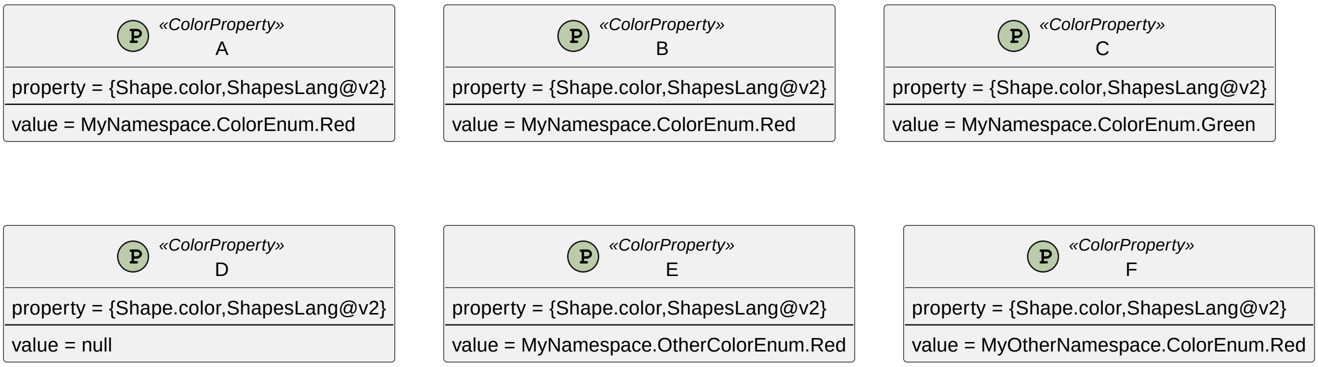
P «ColorProperty» A
property = {Shape.color,ShapesLang@v2}
value = MyNamespace.ColorEnum.Red
P «ColorProperty» B
property = {Shape.color,ShapesLang@v2}
value = MyNamespace.ColorEnum.Red
P «ColorProperty» C
property = {Shape.color,ShapesLang@v2}
value = MyNamespace.ColorEnum.Green
P «ColorProperty» D
property = {Shape.color,ShapesLang@v2}
value = null
P «ColorProperty» E
property = {Shape.color,ShapesLang@v2}
value = MyNamespace.OtherColorEnum.Red
P «ColorProperty» F
property = {Shape.color,ShapesLang@v2}
value = MyOtherNamespace.ColorEnum.Red

*Figure 8. Property enumeration value examples*

- `A` and `B` are equal because they have equal *values*.
- `A` and `C` are not equal because they have different *values* in terms of *enumeration literal*: `Red` vs. `Green`.
- `A` and `D` are not equal because they have different *values*: `Red` vs. null.
- `A` and `E` are not equal because they have different *values* in terms of *enumeration*: `ColorEnum` vs. `OtherColorEnum`.
- `A` and `F` are not equal because they have different *values* in terms of *namespace*: `MyNamespace` vs. `MyOtherNamespace`.

**WARNING**

`E` and `F` should not be possible, because C# types `MyNamespace.ColorEnum`, `MyNamespace.OtherColorEnum`, and `MyOtherNamespace.ColorEnum` should not be compatible. However, due to the way C# implements enumerations, it can happen.

## Reference

We distinguish between *internal* and *external* reference targets. An *internal* target is element of the set of nodes to be compared, an *external* target is not element of this set of nodes. We compare both kinds of targets, but in different ways.

We consider references with *internal* targets equal if their reference and target node match, i.e. their target nodes are considered *comparable*. *Comparable* means they have the same relative position within the compared nodes.

We consider references with *external* targets equal if their reference and target node id match.

**NOTE**

We don't spell out all the differences in reference feature keys in our examples, i.e. *reference.key*, *reference.classifier.key*, *reference.classifier.language.key* and *reference.classifier.language.version*. They apply the same way as for Property features.

**NOTE**

Each box represents one *complete node*.

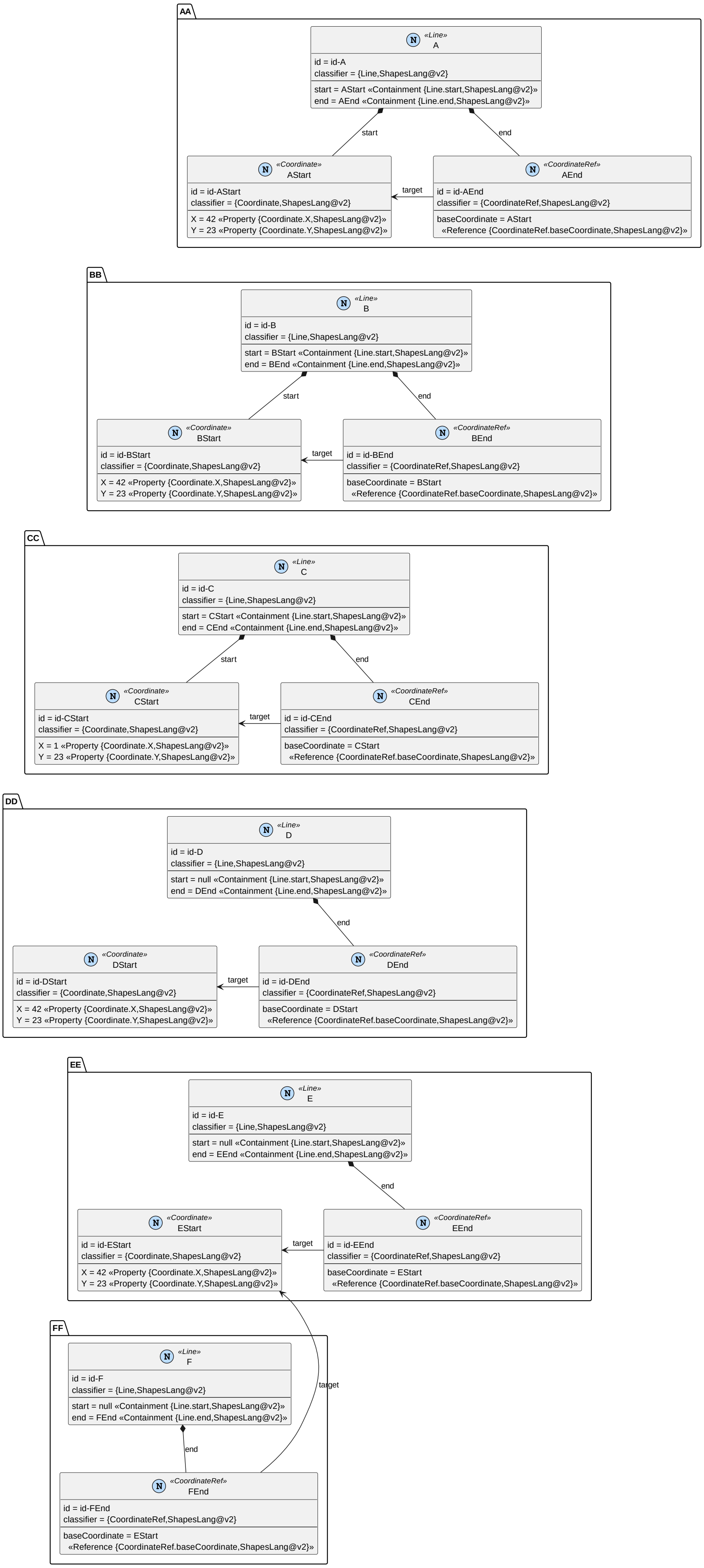
AA
«Line»
A
id = id-A
classifier = {Line,ShapesLang@v2}
start = AStart «Containment {Line.start,ShapesLang@v2}»
end = AEnd «Containment {Line.end,ShapesLang@v2}»
start
end
«Coordinate»
AStart
id = id-AStart
classifier = {Coordinate,ShapesLang@v2}
X = 42 «Property {Coordinate.X,ShapesLang@v2}»
Y = 23 «Property {Coordinate.Y,ShapesLang@v2}»
target
«CoordinateRef»
AEnd
id = id-AEnd
classifier = {CoordinateRef,ShapesLang@v2}
baseCoordinate = AStart
«Reference {CoordinateRef.baseCoordinate,ShapesLang@v2}»
BB
«Line»
B
id = id-B
classifier = {Line,ShapesLang@v2}
start = BStart «Containment {Line.start,ShapesLang@v2}»
end = BEnd «Containment {Line.end,ShapesLang@v2}»
start
end
«Coordinate»
BStart
id = id-BStart
classifier = {Coordinate,ShapesLang@v2}
X = 42 «Property {Coordinate.X,ShapesLang@v2}»
Y = 23 «Property {Coordinate.Y,ShapesLang@v2}»
target
«CoordinateRef»
BEnd
id = id-BEnd
classifier = {CoordinateRef,ShapesLang@v2}
baseCoordinate = BStart
«Reference {CoordinateRef.baseCoordinate,ShapesLang@v2}»
CC
«Line»
C
id = id-C
classifier = {Line,ShapesLang@v2}
start = CStart «Containment {Line.start,ShapesLang@v2}»
end = CEnd «Containment {Line.end,ShapesLang@v2}»
start
end
«Coordinate»
CStart
id = id-CStart
classifier = {Coordinate,ShapesLang@v2}
X = 1 «Property {Coordinate.X,ShapesLang@v2}»
Y = 23 «Property {Coordinate.Y,ShapesLang@v2}»
target
«CoordinateRef»
CEnd
id = id-CEnd
classifier = {CoordinateRef,ShapesLang@v2}
baseCoordinate = CStart
«Reference {CoordinateRef.baseCoordinate,ShapesLang@v2}»
DD
«Line»
D
id = id-D
classifier = {Line,ShapesLang@v2}
start = null «Containment {Line.start,ShapesLang@v2}»
end = DEnd «Containment {Line.end,ShapesLang@v2}»
end
«Coordinate»
DStart
id = id-DStart
classifier = {Coordinate,ShapesLang@v2}
X = 42 «Property {Coordinate.X,ShapesLang@v2}»
Y = 23 «Property {Coordinate.Y,ShapesLang@v2}»
target
«CoordinateRef»
DEnd
id = id-DEnd
classifier = {CoordinateRef,ShapesLang@v2}
baseCoordinate = DStart
«Reference {CoordinateRef.baseCoordinate,ShapesLang@v2}»
EE
«Line»
E
id = id-E
classifier = {Line,ShapesLang@v2}
start = null «Containment {Line.start,ShapesLang@v2}»
end = EEnd «Containment {Line.end,ShapesLang@v2}»
end
«Coordinate»
EStart
id = id-EStart
classifier = {Coordinate,ShapesLang@v2}
X = 42 «Property {Coordinate.X,ShapesLang@v2}»
Y = 23 «Property {Coordinate.Y,ShapesLang@v2}»
target
«CoordinateRef»
EEnd
id = id-EEnd
classifier = {CoordinateRef,ShapesLang@v2}
baseCoordinate = EStart
«Reference {CoordinateRef.baseCoordinate,ShapesLang@v2}»
FF
«Line»
F
id = id-F
classifier = {Line,ShapesLang@v2}
start = null «Containment {Line.start,ShapesLang@v2}»
end = FEnd «Containment {Line.end,ShapesLang@v2}»
target
end
«CoordinateRef»
FEnd
id = id-FEnd
classifier = {CoordinateRef,ShapesLang@v2}
baseCoordinate = EStart
«Reference {CoordinateRef.baseCoordinate,ShapesLang@v2}»

*Figure 9. Reference examples*

Assume we compare `A` and `B`. Implicitly, we also compare all their *containments*, so we actually compare `[A, AStart, AEnd]` vs. `[B, BStart, BEnd]`.
`AEnd.baseCoordinate` and `BEnd.baseCoordinate` are equal because their *reference* match; their *target* is part of the comparison, so they are *internal*; and the targets are considered *comparable*: both their relative position is `root.start`.

Assume we compare `A` and `C`. Implicitly, we also compare all their *containments*, so we actually compare `[A, AStart, AEnd]` vs. `[C, CStart, CEnd]`. Note that `AStart.X` and `CStart.X` have different values (`42` vs. `1`).
`AEnd.baseCoordinate` and `CEnd.baseCoordinate` are equal because their *reference* match; their *target* is part of the comparison, so they are *internal*; and the targets are considered *comparable*: both their relative position is `root.start`. It doesn't matter that `AStart` is not equal to `CStart`.

Assume we compare `A` and `D`. Implicitly, we also compare all their *containments*, so we actually compare `[A, AEnd, AStart]` vs. `[D, DEnd]`.
`AEnd.baseCoordinate` and `DEnd.baseCoordinate` are not equal because their kind doesn't match: *internal* vs. *external*.

Assume we compare `[A, AStart]` and `[D, DStart]`. Implicitly, we also compare all their *containments*, so we actually compare `[A, AEnd, AStart]` vs. `[D, DEnd, DStart]`.
`AEnd.baseCoordinate` and `DEnd.baseCoordinate` are not equal because their *target* does not match: both targets are part of the comparison, so they are *internal*. However, their relative position is different: `root.start` vs. `root`.

Assume we compare `[D, DStart]` and `[E, EStart]`. Implicitly, we also compare all their *containments*, so we actually compare `[D, DEnd, DStart]` vs. `[E, EEnd, EStart]`.
`DEnd.baseCoordinate` and `EEnd.baseCoordinate` are equal because their *reference* match; their *target* is part of the comparison, so they are *internal*; and the targets are considered *comparable*: both their relative position is `root`.

Assume we compare `D` and `E`. Implicitly, we also compare all their *containments*, so we actually compare `[D, DEnd]` vs. `[E, EEnd]`.
`DEnd.baseCoordinate` and `EEnd.baseCoordinate` are not equal because their *target* does not match: both targets are outside the comparison, so they are *external*. However, their *ids* differ: `id-DStart` vs. `id-EStart`.

Assume we compare `E` and `F`. Implicitly, we also compare all their *containments*, so we actually compare `[E, EEnd]` vs. `[F, FEnd]`.
`EEnd.baseCoordinate` and `FEnd.baseCoordinate` are equal because their *reference* match; their *target* is outside the comparison, so they are *external*; and the targets are *equal* because they have equal *ids*.

## Node

We consider nodes equal if their classifier, all their annotations, and all their features match. We consider features matching if equal features are set, and each set feature is equal.

**NOTE** Each box represents one *complete node*.

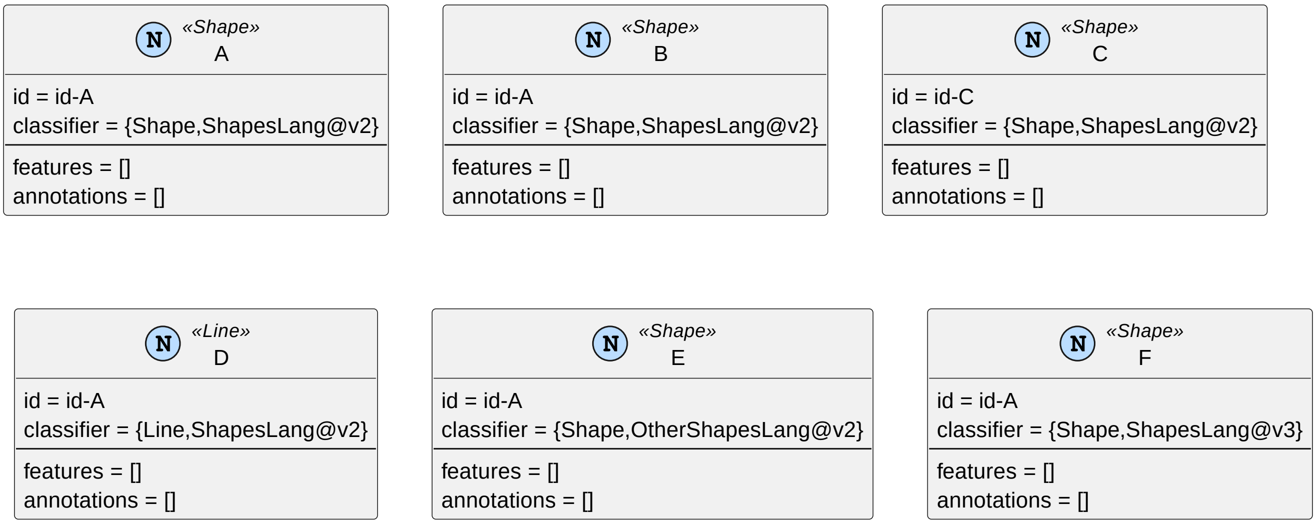

*Figure 10. Feature-less node examples*

- `A` and `B` are equal because they have equal *classifiers*, both no *features*, and both no *annotations*. We don't compare their *ids*.
- `A` and `C` are equal because they have equal *classifiers*, both no *features*, and both no *annotations*. We don't compare their *ids*.
- `A` and `D` are not equal because they have different *classifiers* (in terms of *key*: `Shape` vs. `Line`).
- `A` and `E` are not equal because they have different *classifiers* (in terms of *language key*: `ShapesLang` vs. `OtherShapesLang`).
- `A` and `F` are not equal because they have different *classifiers* (in terms of *language version*: `2` vs. `3`).

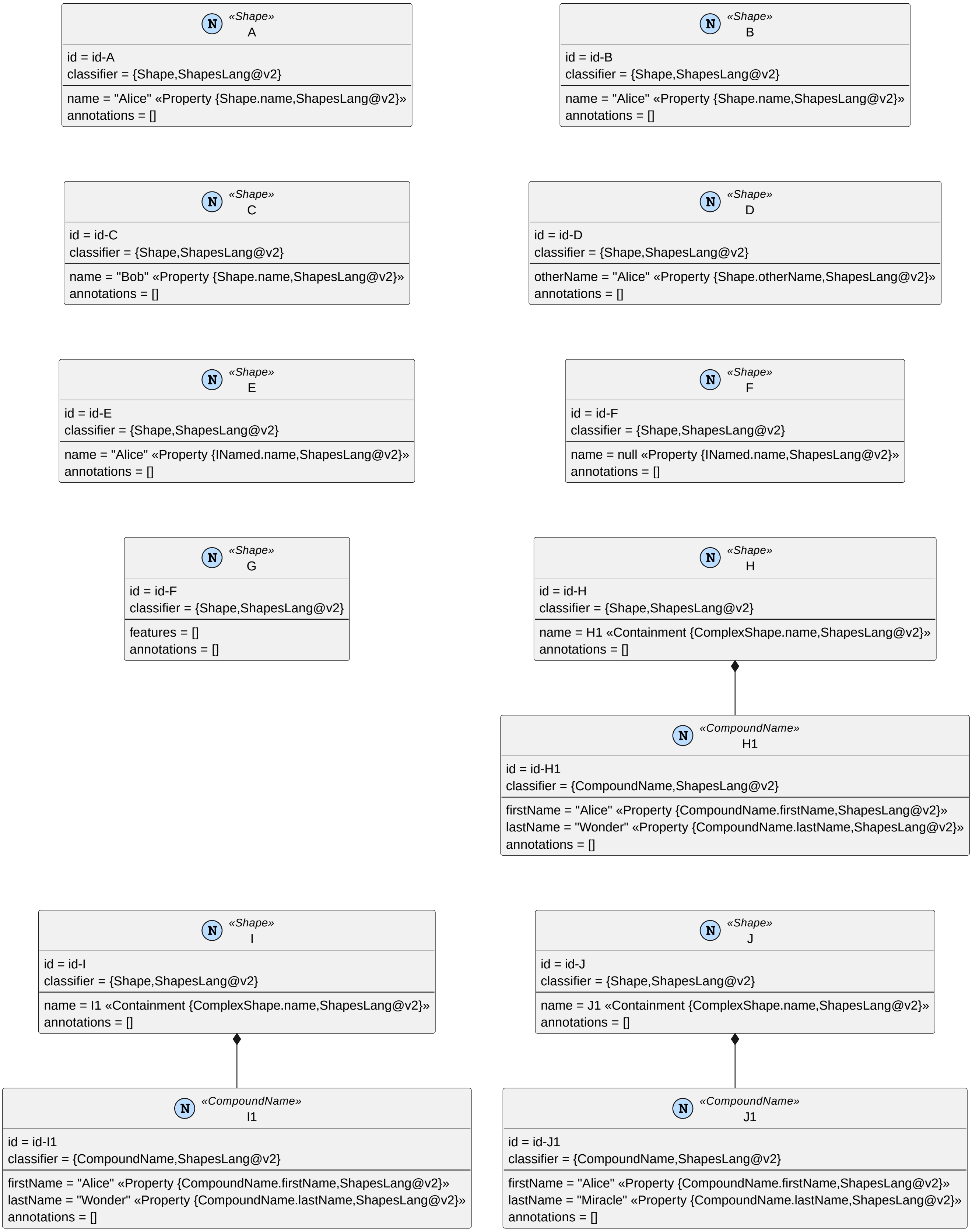
N «Shape» A
id = id-A
classifier = {Shape,ShapesLang@v2}
name = "Alice" «Property {Shape.name,ShapesLang@v2}»
annotations = []
N «Shape» B
id = id-B
classifier = {Shape,ShapesLang@v2}
name = "Alice" «Property {Shape.name,ShapesLang@v2}»
annotations = []
N «Shape» C
id = id-C
classifier = {Shape,ShapesLang@v2}
name = "Bob" «Property {Shape.name,ShapesLang@v2}»
annotations = []
N «Shape» D
id = id-D
classifier = {Shape,ShapesLang@v2}
otherName = "Alice" «Property {Shape.otherName,ShapesLang@v2}»
annotations = []
N «Shape» E
id = id-E
classifier = {Shape,ShapesLang@v2}
name = "Alice" «Property {INamed.name,ShapesLang@v2}»
annotations = []
N «Shape» F
id = id-F
classifier = {Shape,ShapesLang@v2}
name = null «Property {INamed.name,ShapesLang@v2}»
annotations = []
N «Shape» G
id = id-F
classifier = {Shape,ShapesLang@v2}
features = []
annotations = []
N «Shape» H
id = id-H
classifier = {Shape,ShapesLang@v2}
name = H1 «Containment {ComplexShape.name,ShapesLang@v2}»
annotations = []
N «CompoundName» H1
id = id-H1
classifier = {CompoundName,ShapesLang@v2}
firstName = "Alice" «Property {CompoundName.firstName,ShapesLang@v2}»
lastName = "Wonder" «Property {CompoundName.lastName,ShapesLang@v2}»
annotations = []
N «Shape» I
id = id-I
classifier = {Shape,ShapesLang@v2}
name = I1 «Containment {ComplexShape.name,ShapesLang@v2}»
annotations = []
N «Shape» J
id = id-J
classifier = {Shape,ShapesLang@v2}
name = J1 «Containment {ComplexShape.name,ShapesLang@v2}»
annotations = []
N «CompoundName» I1
id = id-I1
classifier = {CompoundName,ShapesLang@v2}
firstName = "Alice" «Property {CompoundName.firstName,ShapesLang@v2}»
lastName = "Wonder" «Property {CompoundName.lastName,ShapesLang@v2}»
annotations = []
N «CompoundName» J1
id = id-J1
classifier = {CompoundName,ShapesLang@v2}
firstName = "Alice" «Property {CompoundName.firstName,ShapesLang@v2}»
lastName = "Miracle" «Property {CompoundName.lastName,ShapesLang@v2}»
annotations = []

*Figure 11. Nodes with features examples*

- `A` and `B` are equal because they have equal *classifiers*, equal *features*, and both no *annotations*.
- `A` and `C` are not equal because they have different *features* (in terms of `name` properties' *value*: `Alice` vs. `Bob`).
- `A` and `D` are not equal because they have different *features*: `name` vs. `otherName`.
- `A` and `E` are not equal because they have different *features*: `Shape.name` vs. `INamed.name`.

- `A` and `F` are not equal because they have different *features*: `name` property *value* `Alice` vs. null.
- `A` and `G` are not equal because they have different *features*: `name` property present vs. no features.
- `A` and `H` are not equal because they have different *features*: `name` property vs. `name` containment.
- `F` and `G` are equal because they have equal *classifiers*, both no annotations, and *semantically* equal `name` property: We don’t distinguish between an *unset* feature and a feature with null value (or empty list, in case of multi-value feature).
- `H` and `I` are equal because they have equal *classifiers*, both no annotations, and both equal `name` feature (equal contained `name` nodes).
- `H` and `J` are not equal because they have different *features*: contained `name` nodes are not equal (in terms of property `lastName`: `Wonder` vs. `Miracle`).

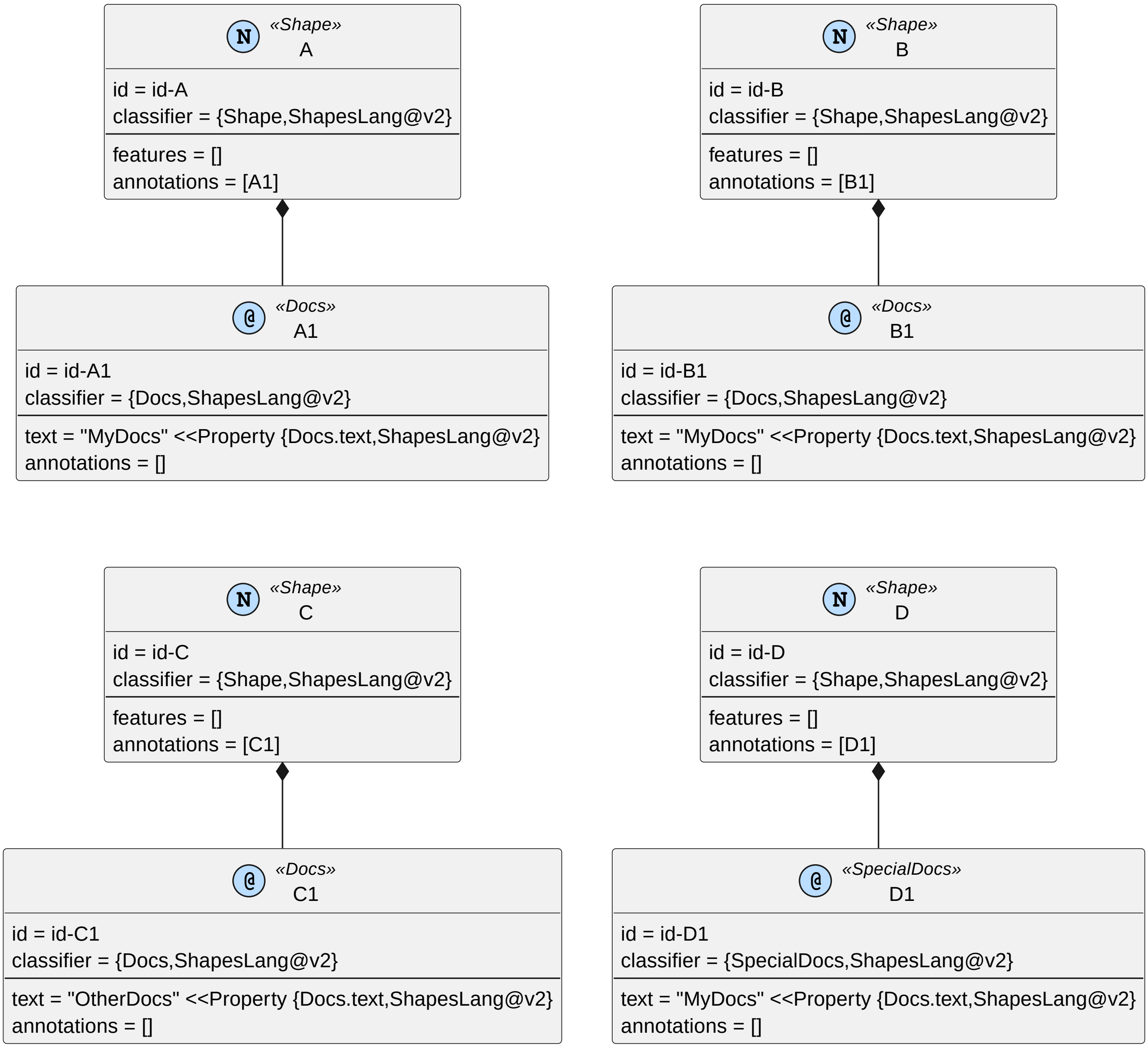
N «Shape» A
id = id-A
classifier = {Shape,ShapesLang@v2}
features = []
annotations = [A1]
@ «Docs» A1
id = id-A1
classifier = {Docs,ShapesLang@v2}
text = "MyDocs" <<Property {Docs.text,ShapesLang@v2}
annotations = []
N «Shape» B
id = id-B
classifier = {Shape,ShapesLang@v2}
features = []
annotations = [B1]
@ «Docs» B1
id = id-B1
classifier = {Docs,ShapesLang@v2}
text = "MyDocs" <<Property {Docs.text,ShapesLang@v2}
annotations = []
N «Shape» C
id = id-C
classifier = {Shape,ShapesLang@v2}
features = []
annotations = [C1]
@ «Docs» C1
id = id-C1
classifier = {Docs,ShapesLang@v2}
text = "OtherDocs" <<Property {Docs.text,ShapesLang@v2}
annotations = []
N «Shape» D
id = id-D
classifier = {Shape,ShapesLang@v2}
features = []
annotations = [D1]
@ «SpecialDocs» D1
id = id-D1
classifier = {SpecialDocs,ShapesLang@v2}
text = "MyDocs" <<Property {Docs.text,ShapesLang@v2}
annotations = []

*Figure 12. Nodes with annotations examples*

- `A` and `B` are equal because they have equal *classifiers*, both no features, and both equal annotations (equal annotation nodes).
- `A` and `C` are not equal because they have different *annotations*: annotation nodes differ in terms of `text` property: `MyDocs` vs. `OtherDocs`.

- `A` and `D` are not equal because they have different *annotations*: annotation nodes differ in terms of *classifier key*: `Docs` vs. `SpecialDocs`.

# Nodes (list of nodes)

Node lists may appear at different places:

- Root-level parameter to Comparer.
- List of node annotations.
- Multi-valued containment.

We consider lists of nodes equal if they have the same length, and the nodes at each position are equal.

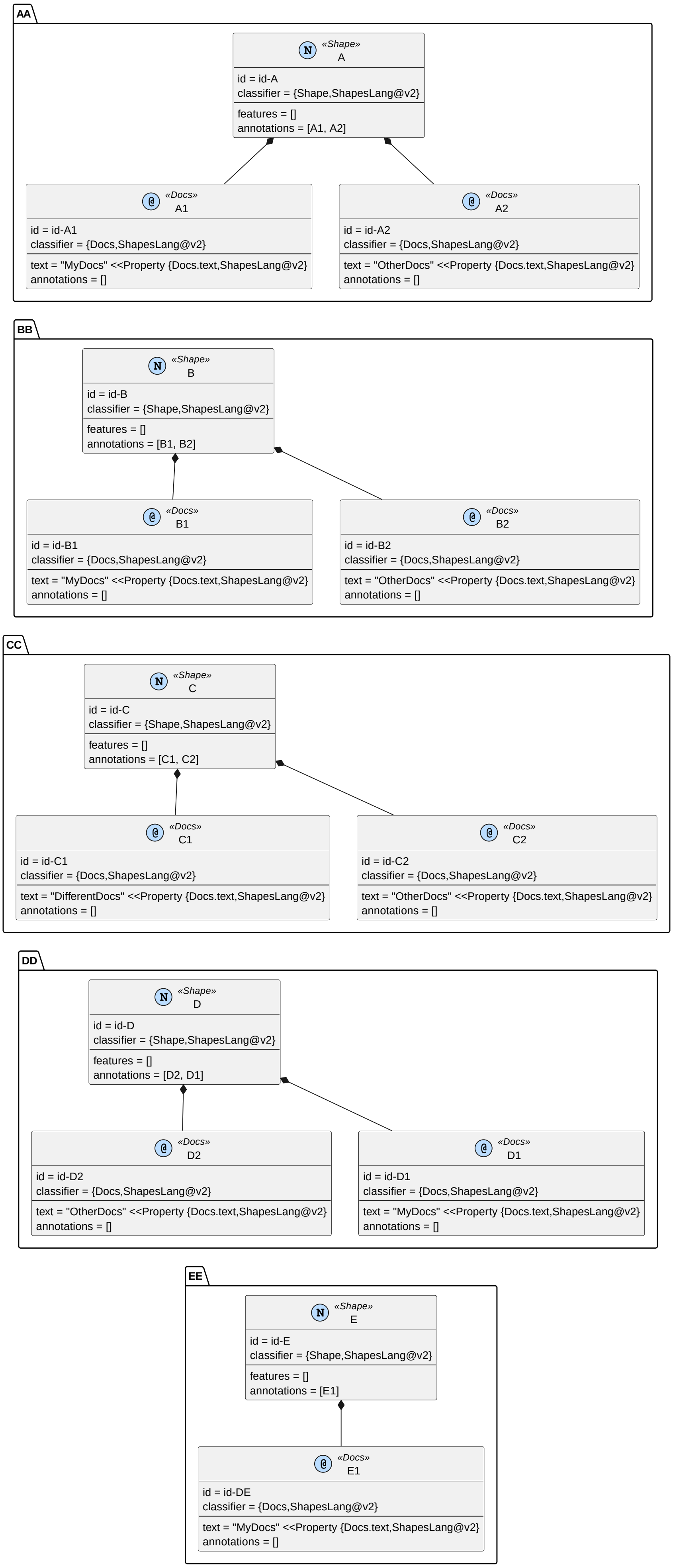
AA
N «Shape» A
id = id-A
classifier = {Shape,ShapesLang@v2}
features = []
annotations = [A1, A2]
@ «Docs» A1
id = id-A1
classifier = {Docs,ShapesLang@v2}
text = "MyDocs" <<Property {Docs.text,ShapesLang@v2}
annotations = []
@ «Docs» A2
id = id-A2
classifier = {Docs,ShapesLang@v2}
text = "OtherDocs" <<Property {Docs.text,ShapesLang@v2}
annotations = []
BB
N «Shape» B
id = id-B
classifier = {Shape,ShapesLang@v2}
features = []
annotations = [B1, B2]
@ «Docs» B1
id = id-B1
classifier = {Docs,ShapesLang@v2}
text = "MyDocs" <<Property {Docs.text,ShapesLang@v2}
annotations = []
@ «Docs» B2
id = id-B2
classifier = {Docs,ShapesLang@v2}
text = "OtherDocs" <<Property {Docs.text,ShapesLang@v2}
annotations = []
CC
N «Shape» C
id = id-C
classifier = {Shape,ShapesLang@v2}
features = []
annotations = [C1, C2]
@ «Docs» C1
id = id-C1
classifier = {Docs,ShapesLang@v2}
text = "DifferentDocs" <<Property {Docs.text,ShapesLang@v2}
annotations = []
@ «Docs» C2
id = id-C2
classifier = {Docs,ShapesLang@v2}
text = "OtherDocs" <<Property {Docs.text,ShapesLang@v2}
annotations = []
DD
N «Shape» D
id = id-D
classifier = {Shape,ShapesLang@v2}
features = []
annotations = [D2, D1]
@ «Docs» D2
id = id-D2
classifier = {Docs,ShapesLang@v2}
text = "OtherDocs" <<Property {Docs.text,ShapesLang@v2}
annotations = []
@ «Docs» D1
id = id-D1
classifier = {Docs,ShapesLang@v2}
text = "MyDocs" <<Property {Docs.text,ShapesLang@v2}
annotations = []
EE
N «Shape» E
id = id-E
classifier = {Shape,ShapesLang@v2}
features = []
annotations = [E1]
@ «Docs» E1
id = id-DE
classifier = {Docs,ShapesLang@v2}
text = "MyDocs" <<Property {Docs.text,ShapesLang@v2}
annotations = []

*Figure 13. Node list examples*

- `A.annotations` and `B.annotations` are equal because they have the same length (`2`) and the nodes at each position are equal (`A1` at position 0 is equal to `B1` at position 0, and `A2` at position 1 is equal to `B2` at position 1).
- `A.annotations` and `C.annotations` are not equal because position 0 differs in terms of the node's `text` property: `MyDocs` vs. `DifferentDocs`.
- `A.annotations` and `D.annotations` are not equal because position 0 and 1 are flipped: `[A1, A2]` vs. `[D2, D1]`. (Technically, we just compare `A1` to `D2` and they differ in terms of their `text` property.)
- `A.annotation` and `E.annotation` are not equal because of their size: `2` vs. `1`.

## References (list of references)

Reference features may be multi-valued.

We consider lists of references equal if they have the same length, and the references at each position are equal.