49. Notation

Description

This chapter explains the notation used in this book.

Program text

The program examples shown in this book is written in the qBeta language. The elements of the language are introduced as they are needed to illustrate the programming elements presented in the book. For a comprehensive description of qBeta see the web-site qbeta.dev.

Abbreviations

For practical purposes, we do not always show all the code in the examples and we use the following notation for code not shown:

- The symbols `-"-` stands for code, which has been shown in a previous example. You may think of the `-"-` as an extended ditto-sign.
- The symbols `:::` stands for code, which will be shown later – often in the same section.
- The symbols `...` stands for code, which is not shown and left to the reader to fill in.

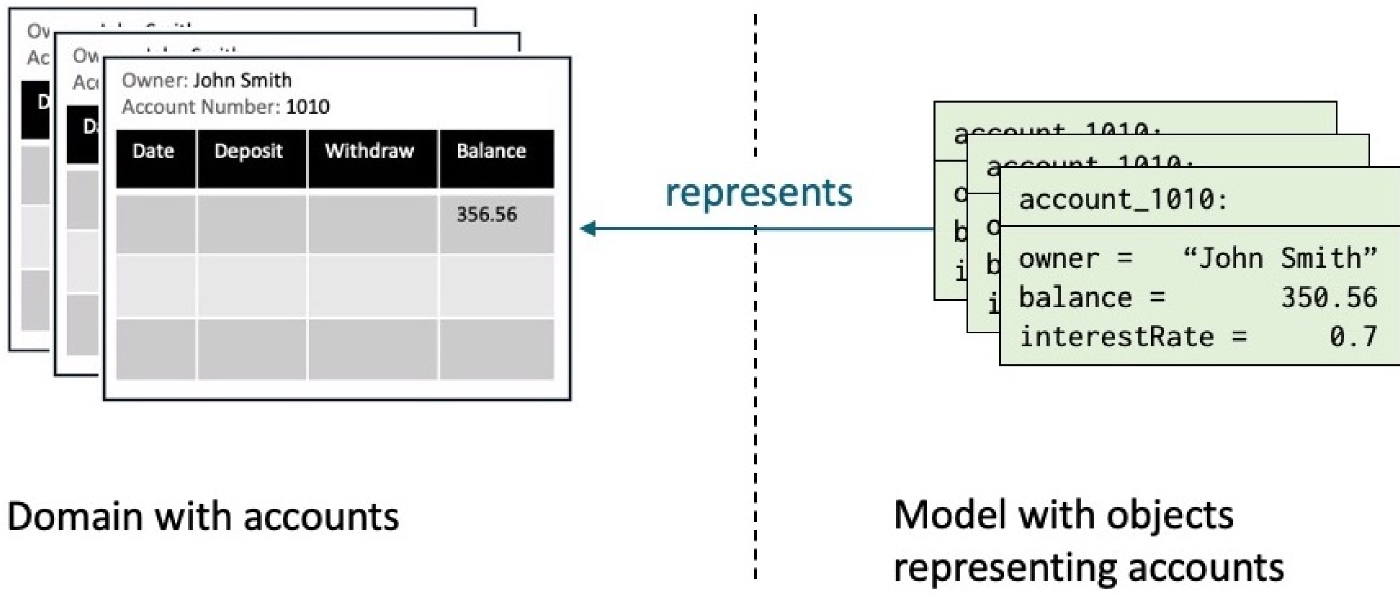
Diagrams and illustrations

In this section, we summative the *diagrams* used to show different kinds of elements of programs and program executions. Diagrams are partial since they only show selected elements of a given program or program execution, but they are formal/exact/precise in the sense that they show elements that are present in the program or program execution.

We also use *illustrations* in the form of informal drawings or pictures to illustrate a given text in the book, a given application domain, etc.

2024-11-01: Her skal være et foto, eller tegning af et domæne, eller lign. uden objekter

We use a combination of pictures and diagrams to show the relationships between a domain and a model.



In the following sections, we present the diagrams being used.

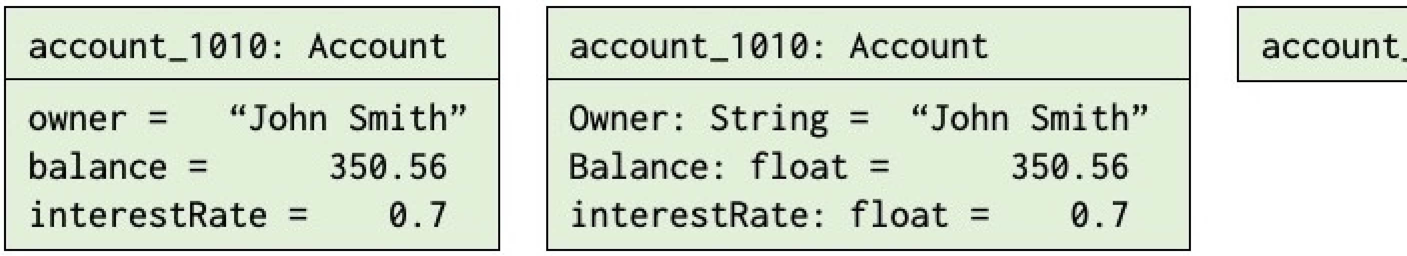
Objects

Class-defined and singular objects

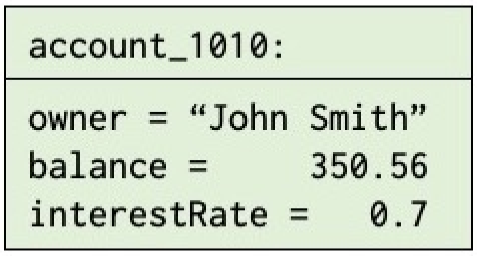
Rectangles of the following forms are used to illustrate objects.

Class-defined object

The notation for an object of a class is a rectangle including an upper part with the name of the object followed by the name of the class, and an optional lower part with a selected set, which may be empty, of the attributes. A data-item always includes name and value, while its type may be omitted.



Singular object



The notation for a singular object is a rectangle with an upper part with just the name of the object, and an optional lower part with the same contents as a class defined object.

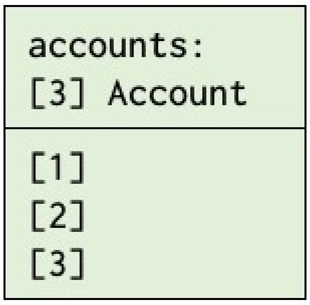
Common to both is

- The name of the object is left-justified and followed by a ':'.
- The color of the background is light green.

Array objects

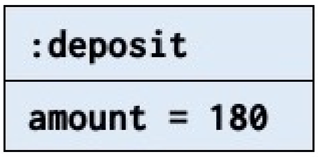
The notation for an array object is a rectangle with an upper part with the name of the array object, the number of elements in the array and the type (class) of the elements in the array. The lower part is a list of the elements in the array. This list may be partial and only show selected elements of the array.

Vi mangler et eksempel der er fyldt ud med nogle elementer . Det har vi i kap 3, så det skal jeg sætte ind.



Method objects

The notation for a method object is a rectangle with an upper part with the name of the method, and a lower part with a selected set of its attributes. A data-item always includes name and value, while its type may be omitted.



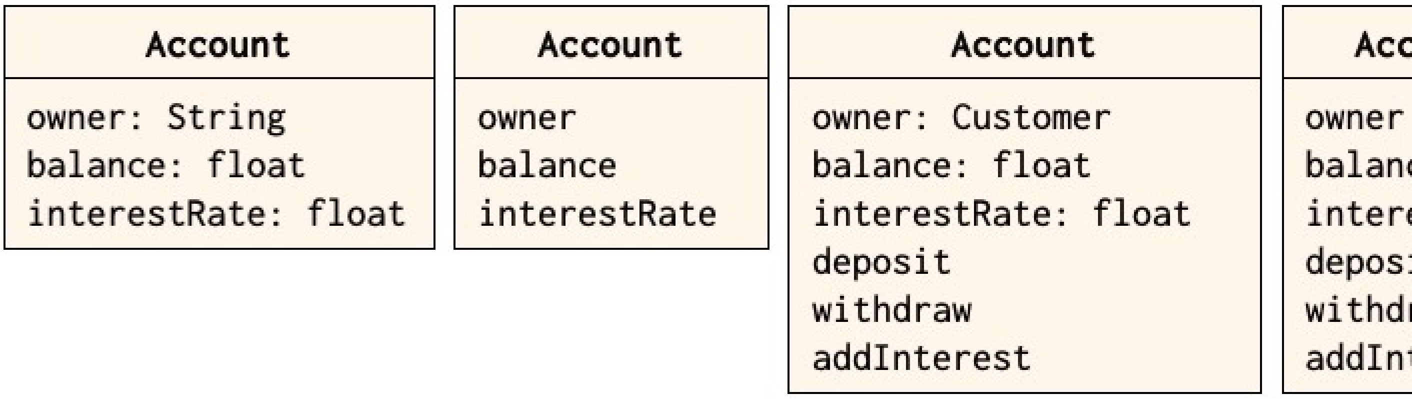
Der skal vel også være en for singular method object?
`cycle`
`v := v + 1`
Debuggeren viser en boks med navnet `cycle$112` – er det så `cycle<`

Singular method objects

+++++

Classes

A rectangle of the following forms is used to illustrate a class.



The notation for illustrating a class is a rectangle including an upper part with the name of the class, and a part with a selected set of its attributes.

- A data-item always includes name, while its type may be omitted.
- A method-item just include its name.

The name of the class is in boldface and centered.

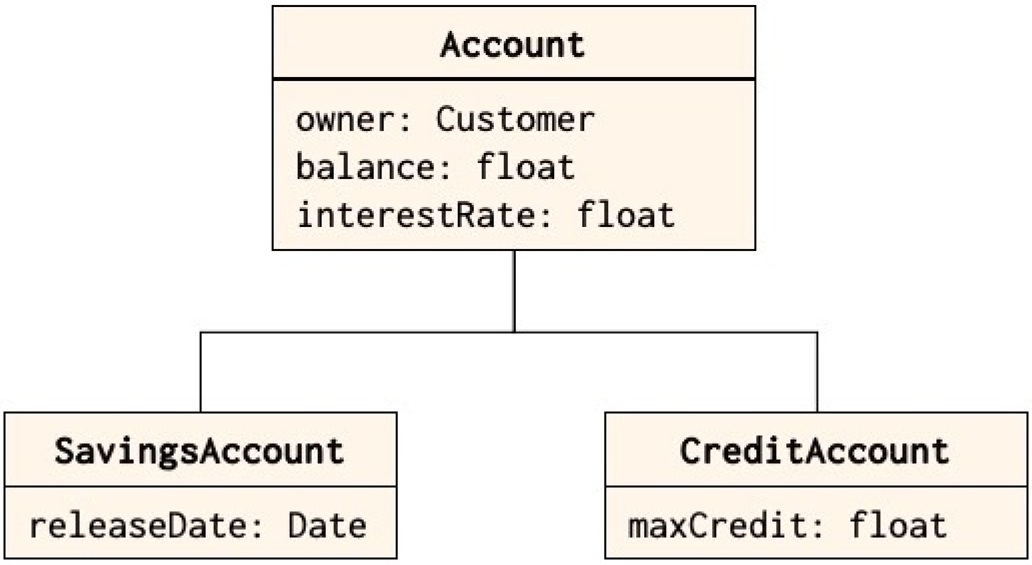
The color of the background for both compartment is light yellow.

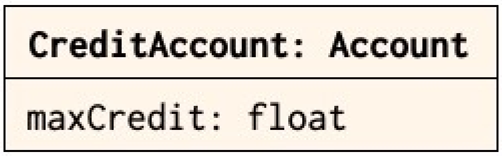
No that it differs from the way objects are illustrated.

Subclass

The notation for a class being a subclass of another class is a line from the subclass to the superclass, with the subclass below the superclass.

A class may have an arbitrary number of direct subclasses and a subclass may itself have subclasses. The subclass hierarchy may thus have an arbitrary depth. +++ måske vi bør have et ex med flere niveauer i hierarkiet?



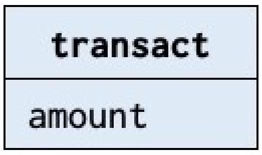


Alternatively the name of the superclass may be placed after the ':' in a class.

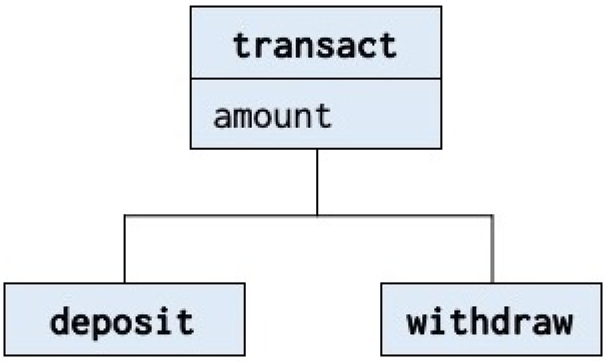
Methods/submethods

Method

The notation for illustrating a method is similar to the notation for classes, however with different color, that is light blue. The upper part holds the name of the method and the lower part holds attributes, including parameters.



Submethod



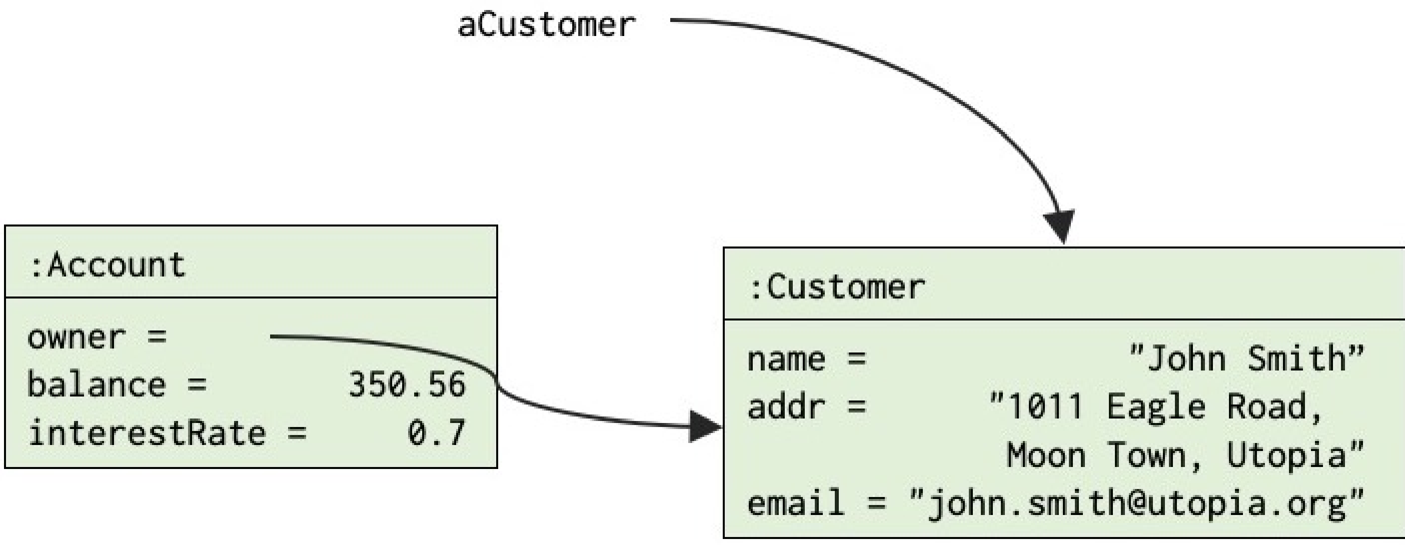
The notation for a method that is a submethod of another method is the same as for subclasses: a line from the submethod to the supermethod, with a closed arrow directed towards the supermethod.

As for subclasses, a method may have an arbitrary number of submethods and the sub method hierarchy may have an arbitrary depth.

References

A reference its shown as an arrow pointing from a reference data-item to the object being referred to. Sometimes this data-item is shown as an attribute of an object and sometimes it is just shown on its own and not as an attribute of an object.

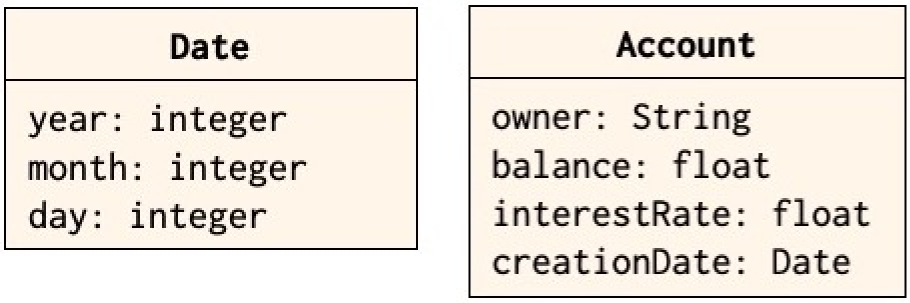
+++Så da behøver vi egentlig ikke demo/helper objekter



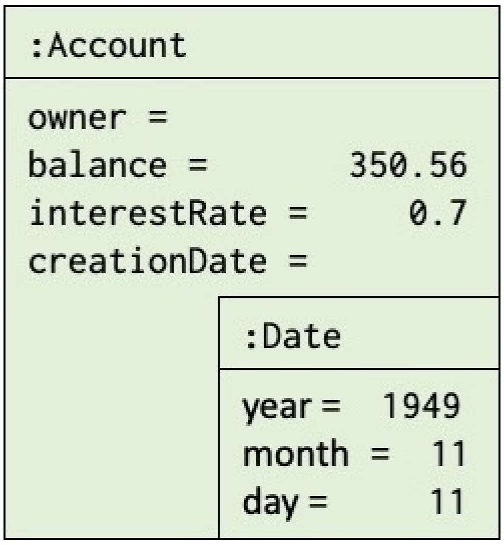
Composite value type

Definition and typing an attribute by a composite type.

Kan egentlig ikke se at Date er en value type. Subklasse til Value?



+++ corresponding object



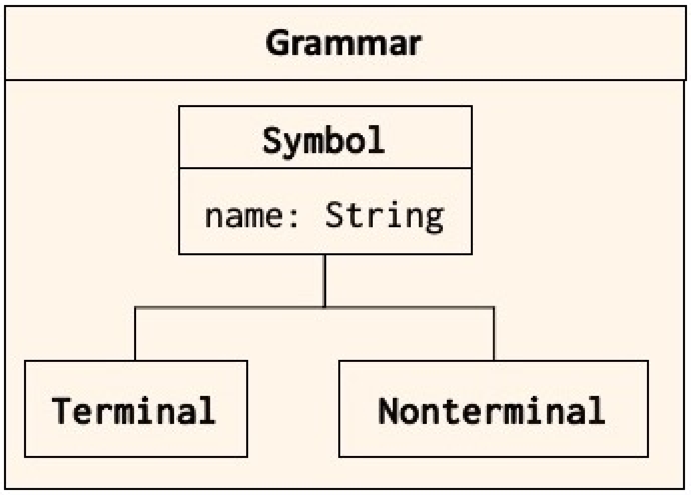
Nesting

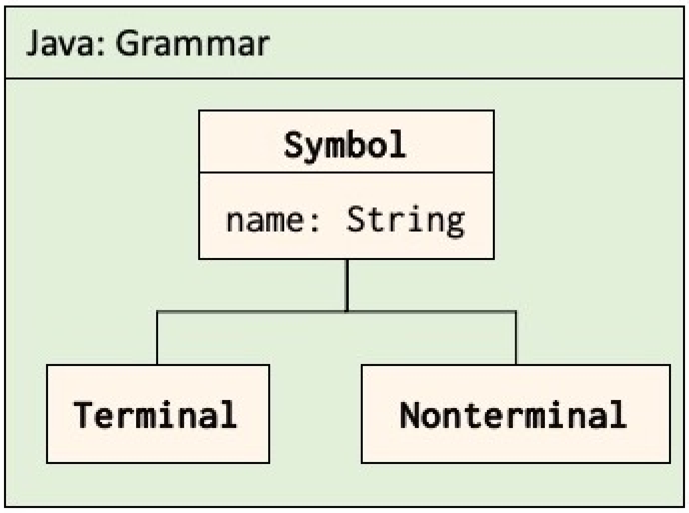
For illustrating nesting we have two alternatives:

The **first alternative** is to include classes in the lower part of a class or an object.

The uppermost notation shows that the class `Grammar` has nested classes `Symbol`, `Terminal`, and `NonTerminal`, in addition that `Terminal`, and `NonTerminal` are subclasses of `Symbol`.

The lowermost notation shows that for each instance of class `Grammar` (in this case Java) has nested classes `Symbol`, `Terminal`, and `NonTerminal`. These classes are different from `Symbol`, `Terminal`, and `NonTerminal` in some other instance of `Grammar`.

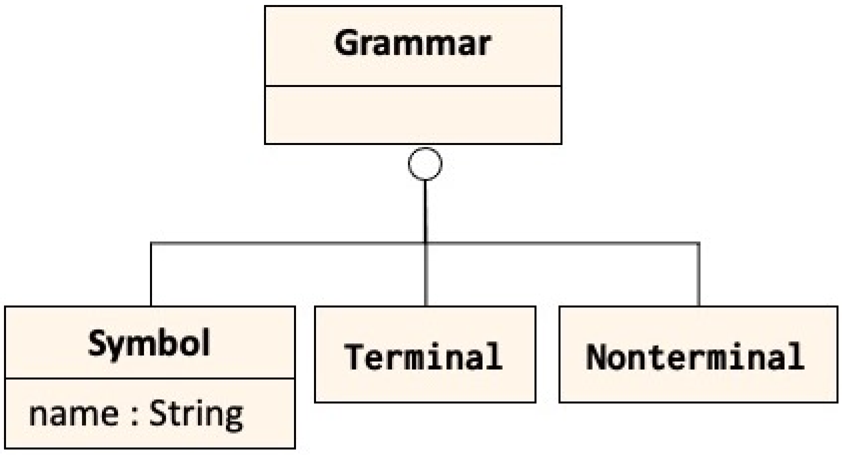


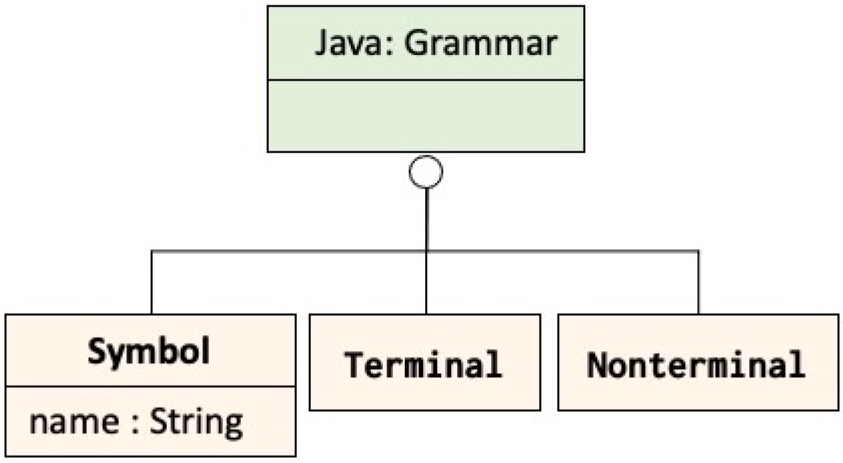


The **second alternative** for showing nesting is to have the nested classes connected to the enclosing class or object, ending with a small circle.

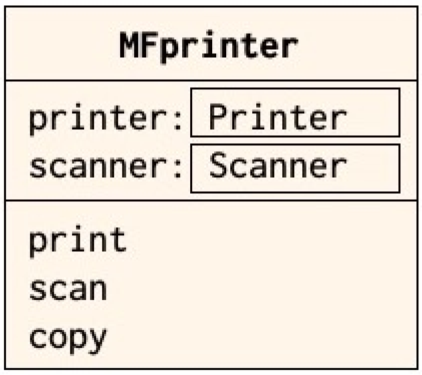
The uppermost notation shows that the class `Grammar` has nested classes `Symbol`, `Terminal`, and `NonTerminal`.

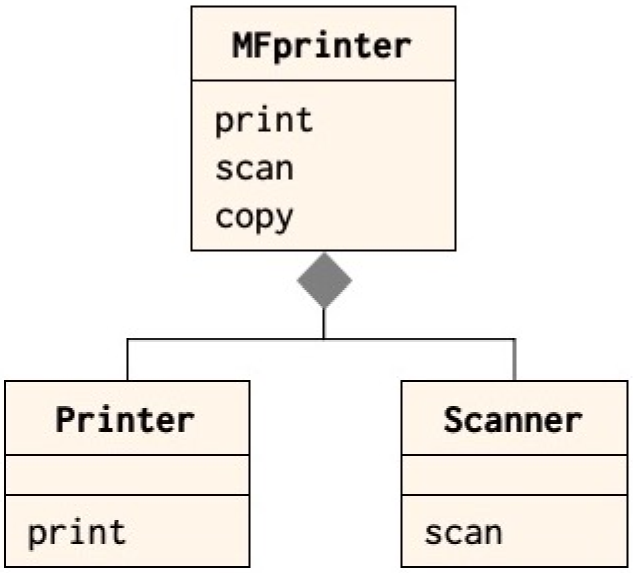
The lowermost notation shows that for each instance of class `Grammar` (in this case Java) has nested classes `Symbol`, `Terminal`, and `NonTerminal`. These classes are different from `Symbol`, `Terminal`, and `NonTermal` in some other instance of `Grammar`.





Composition, part objects

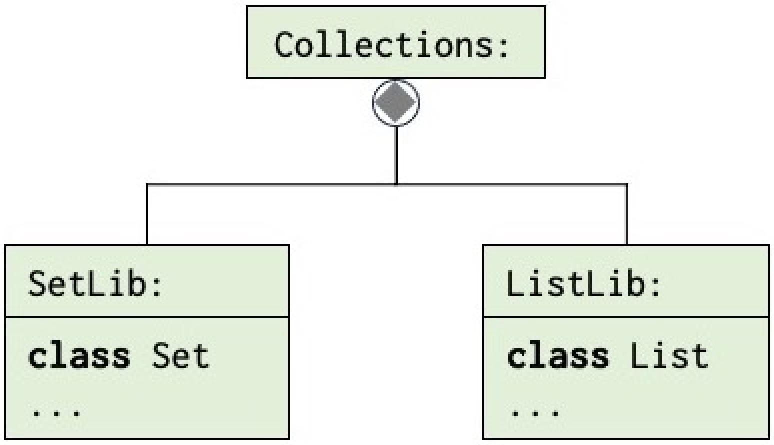




Definition of class with part-objects, two alternatives

Combined nesting and composition

The combination of nesting and composition is shown by a combination of the symbols for nesting and composition. This is typically used to illustrate that singular objects are described in the context of an object or class.



Object sequence diagrams

…