# Create your code generator with BESSER

## Welcome to our BESSER lab guide!

In this guide, you will learn to use BESSER from the perspective of a developer user, specifically for developing a code generator.

## Context

As you've seen in the previous guide, BESSER provides the B-UML modeling language for creating different types of models. In this case, we will develop a code generator that takes as input a structural model defined with B-UML and generates Java code representing the object model, i.e., the classes and attributes in the Java language.

To create the code generator, you can utilize the interface provided by BESSER and define a Jinja[[1]](#footnote-1) template for code generation. You can find more information about this in [this BESSER documentation](https://besser.readthedocs.io/en/latest/generators/build_generator.html).

## Creating your Code Generator in BESSER

To create your code generator, you should create a class (for example, JavaGenerator) using the *GeneratorInterface* interface provided by BESSER, ensuring that all BESSER code generators maintain a consistent structure. You can use the following code to create your Java code generator. The constructor method will receive the *DomainModel* or B-UML model as an input parameter, while the *generate()* method performs the code generation. Note that the ***java\_template.py.j2*** template is used for code generation.

Copy this code in a new file named *java\_generator.py*

**import** os

**from** jinja2 **import** Environment**,** FileSystemLoader

**from** besser.BUML.metamodel.structural **import** DomainModel

**from** besser.generators **import** GeneratorInterface

**class** JavaGenerator**(**GeneratorInterface**):**

**def** \_\_init\_\_**(**self**,** model**:** DomainModel**,** output\_dir**:** str **=** **None):**

super**().**\_\_init\_\_**(**model**,** output\_dir**)**

**def** generate**(**self**):**

file\_path **=** self**.**build\_generation\_path**(**file\_name**=**"code.java"**)**

templates\_path **=** os**.**path**.**join**(**os**.**path**.**dirname**(**

os**.**path**.**abspath**(**\_\_file\_\_**)),** "templates"**)**

env **=** Environment**(**loader**=**FileSystemLoader**(**

templates\_path**),** trim\_blocks**=True,** lstrip\_blocks**=True,** extensions**=[**'jinja2.ext.do'**])**

template **=** env**.**get\_template**(**'java\_template.py.j2'**)**

**with** open**(**file\_path**,** mode**=**"w"**)** **as** f**:**

generated\_code **=** template**.**render**(**model**=**self**.**model**)**

f**.**write**(**generated\_code**)**

print**(**"Code generated in the location: " **+** file\_path**)**

## Jinja template example

Let's create an initial example Jinja template. To do this, create a file named ***templates/java\_template.py.j2*** and write the following code.

This is a template example to list the name of the classes

**{%** **for** **class** in model**.**get\_classes**()** **%}**

**class** **{{** class**.**name **}}**

**{%** endfor **%}**

Now let's test the code generator. First, we need to create a B-UML model, and then instantiate the generator to obtain the generated code. Execute the following code and check the code generated in the ***output*** folder.

**from** besser.BUML.metamodel.structural **import** DomainModel

**from** besser.BUML.metamodel.structural **import** DomainModel**,** Class**,** Property**,** \

PrimitiveDataType**,** Multiplicity**,** BinaryAssociation

**from** besser.utilities **import** ModelSerializer

**from** java\_generator **import** JavaGenerator

*############################*

*# BUML model definition #*

*############################*

*# Primitive DataTypes*

t\_int**:** PrimitiveDataType **=** PrimitiveDataType**(**"int"**)**

t\_str**:** PrimitiveDataType **=** PrimitiveDataType**(**"str"**)**

t\_datetime**:** PrimitiveDataType **=** PrimitiveDataType**(**"datetime"**)**

*# Library attributes definition*

library\_name**:** Property **=** Property**(**name**=**"name"**,** property\_type**=**t\_str**)**

address**:** Property **=** Property**(**name**=**"address"**,** property\_type**=**t\_str**)**

*# Library class definition*

library**:** Class **=** Class**(**name**=**"Library"**,** attributes**={**library\_name**,** address**})**

*# Book attributes definition*

title**:** Property **=** Property**(**name**=**"title"**,** property\_type**=**t\_str**)**

pages**:** Property **=** Property**(**name**=**"pages"**,** property\_type**=**t\_int**)**

release**:** Property **=** Property**(**name**=**"release"**,** property\_type**=**t\_datetime**)**

*# Book class definition*

book**:** Class **=** Class**(**name**=**"Book"**,** attributes**={**title**,** pages**,** release**})**

*# Author attributes definition*

author\_name**:** Property **=** Property**(**name**=**"name"**,** property\_type**=**t\_str**)**

email**:** Property **=** Property**(**name**=**"email"**,** property\_type**=**t\_str**)**

*# Author class definition*

author**:** Class **=** Class**(**name**=**"Author"**,** attributes**={**author\_name**,** email**})**

*# Library-Book association definition*

located\_in**:** Property **=** Property**(**name**=**"locatedIn"**,** property\_type**=**library**,** multiplicity**=**Multiplicity**(1,** **1))**

has**:** Property **=** Property**(**name**=**"has"**,** property\_type**=**book**,** multiplicity**=**Multiplicity**(0,** "\*"**))**

lib\_book\_association**:** BinaryAssociation **=** BinaryAssociation**(**name**=**"lib\_book\_assoc"**,** ends**={**located\_in**,** has**})**

*# Book-Author association definition*

publishes**:** Property **=** Property**(**name**=**"publishes"**,** property\_type**=**book**,** multiplicity**=**Multiplicity**(0,** "\*"**))**

writed\_by**:** Property **=** Property**(**name**=**"writedBy"**,** property\_type**=**author**,** multiplicity**=**Multiplicity**(1,** "\*"**))**

book\_author\_association**:** BinaryAssociation **=** BinaryAssociation**(**name**=**"book\_author\_assoc"**,** ends**={**writed\_by**,** publishes**})**

*# Domain model definition*

library\_model**:** DomainModel **=** DomainModel**(**name**=**"Library model"**,** types**={**library**,** book**,** author**},**

associations**={**lib\_book\_association**,** book\_author\_association**})**

*############################*

*# Code Generation #*

*############################*

generator**:** JavaGenerator **=** JavaGenerator**(**model**=**library\_model**)**

generator**.**generate**()**

The B-UML model defined in the previous code corresponds to the diagram in Figure 1. Therefore, by running the above code, you should obtain a file containing the names of the three classes (Library, Book, and Author) as output.

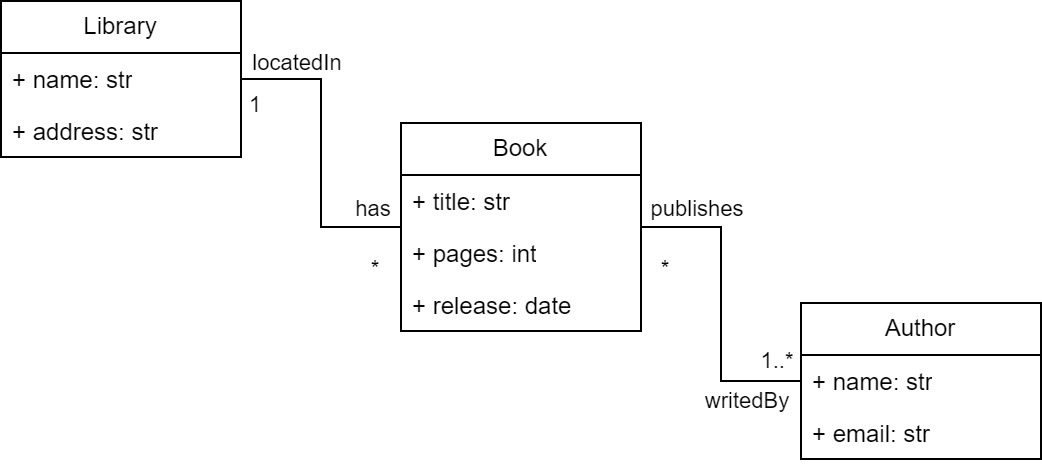


Figure 1. Example model domain

## Exercise

Modify the ***java\_template.py.j2*** to build a Java code generator. In other words, your code generator should produce a set of classes in the Java language with their respective methods, attributes, etc. For example, when providing a model like the one in Figure 1, your code generator should produce a file with the following code.

**import** java.util.List**;**

**import** java.util.ArrayList**;**

**import** java.util.Date**;**

**public** **class** Library **{**

**private** String name**;**

**private** String address**;**

**private** List**<**Book**>** books**;**

**public** Library**(**String name**,** String address**)** **{**

**this.**name **=** name**;**

**this.**address **=** address**;**

**this.**books **=** **new** ArrayList**<>();**

**}**

**public** String getName**()** **{**

**return** name**;**

**}**

**public** **void** setName**(**String name**)** **{**

**this.**name **=** name**;**

**}**

**public** String getAddress**()** **{**

**return** address**;**

**}**

**public** **void** setAddress**(**String address**)** **{**

**this.**address **=** address**;**

**}**

**public** List**<**Book**>** getBooks**()** **{**

**return** books**;**

**}**

**public** **void** addBook**(**Book book**)** **{**

books**.**add**(**book**);**

**}**

**}**

**public** **class** Book **{**

**private** String title**;**

**private** **int** pages**;**

**private** Date release**;**

**private** List**<**Author**>** authors**;**

**public** Book**(**String title**,** **int** pages**,** Date release**)** **{**

**this.**title **=** title**;**

**this.**pages **=** pages**;**

**this.**release **=** release**;**

**this.**authors **=** **new** ArrayList**<>();**

**}**

**public** String getTitle**()** **{**

**return** title**;**

**}**

**public** **void** setTitle**(**String title**)** **{**

**this.**title **=** title**;**

**}**

**public** **int** getPages**()** **{**

**return** pages**;**

**}**

**public** **void** setPages**(int** pages**)** **{**

**this.**pages **=** pages**;**

**}**

**public** Date getRelease**()** **{**

**return** release**;**

**}**

**public** **void** setRelease**(**Date release**)** **{**

**this.**release **=** release**;**

**}**

**public** List**<**Author**>** getAuthors**()** **{**

**return** authors**;**

**}**

**public** **void** addAuthor**(**Author author**)** **{**

authors**.**add**(**author**);**

**}**

**}**

**public** **class** Author **{**

**private** String name**;**

**private** String email**;**

**public** Author**(**String name**,** String email**)** **{**

**this.**name **=** name**;**

**this.**email **=** email**;**

**}**

**public** String getName**()** **{**

**return** name**;**

**}**

**public** **void** setName**(**String name**)** **{**

**this.**name **=** name**;**

**}**

**public** String getEmail**()** **{**

**return** email**;**

**}**

**public** **void** setEmail**(**String email**)** **{**

**this.**email **=** email**;**

**}**

**}**

1. Jinja templates are a way to generate dynamic content in Python-based web applications.

   <https://palletsprojects.com/p/jinja/> [↑](#footnote-ref-1)