Beulah Works LLC

Software Design Document (SDD) Version 1.2

UML Sequence Diagram File Generator

March 6, 2019

Isis Curiel, Bruno Hnatusko III, Brayden Mccoy,

Dhyey Patel, Jesse Primiani, Jacob Taylor,

and Syed Arshiyan Ali Zaidi

Approvals:

Title	**Signature**	**Date**
Project Manager	Isis Curiel	03/22/19
Integration Engineer	Dhyey Patel	03/22/19
Software Architect	Jesse Primiani	03/22/19
Software Architect	Brayden McCoy	03/22/19

Revision History

Date	Revision	Description	Author
03-06-2019	0.01	Created	Brayden McCoy
03-10-2019	0.1	Added basis of document	Brayden McCoy
03-11-2019	0.2	Created Class and Sequence Diagrams	Jesse Primiani
03-17-2019	0.3	Clarified and expanded on the project design	Jesse Primiani
03-20-2019	0.4	Created detailed design, Section 6.1	Brayden McCoy
03-20-2019	0.5	Updated Diagrams and completed Section 5	Jesse Primiani
03-21-2019	1.0	Final tweaks and adjustments as needed	Jesse Primiani
03-30-2019	1.1	Improved design and added more methods	Jesse Primiani
04-21-2019	1.2	Made some slight design improvements	Jesse Primiani

Table of Contents

1 Introduction

1.1. Purpose

This document describes the packages and classes that will compose the UML Sequence Diagram File Generator project. It is also intended to give a detailed description of each component of the software, in order to provide a better understanding of the project's design.

1.2. Scope

This document covers the design of and the classes necessary for the UML Sequence Diagram File Generator project.

1.3. Definitions, Acronyms, Abbreviations

SDD	Software Design Document
UML	Unified Modeling Language
SDM	Sequence Diagram Model
SRS	Software Requirements Specifications
VSDX	Visio file extension

2 References

- This project's Software Requirement Specification document
- IEEE Standard 1016-1993

3 Decomposition Description

The architecture is described using a class model. A class diagram describing aspects of this project's architecture is given in the appendix under "Architecture Class Diagram". The sequence diagram entitled "Setup Sequence Diagram" gives a good overview on the usage of this package. Supporting use case and state models are given in the SRS. All components responsible for converting a Sequence Diagram given in a UMLSequenceDiagram object to a file are located in the package: com.beulahworks.SDMfileGenerator.

3.1. Module Decomposition

This project consists of the *SDMfileGenerator* package. The classes within the package are briefly explained below. Unless stated otherwise, all classes within the package are public.

3.1.1. SDMfileGenerator Package

This package consists of the classes handling the process of taking an SDM object and exporting it to a file. The primary component is the SDMtoFile class (SRS, section 3.2.1.), which is responsible for configuring the output file's name, location, and type, as well as specifying which output and input adapters to use. It performs the actual conversion from the input adapter's data structures to the output adapter's data structures before the output file is created as well. Custom exceptions use the SDMException class.

The OutputType abstract class contains information on the output file extension and the output file type. For specific file types, it is extended with concrete classes, preferably by appending the library name and type to the class name, as in OutputTypeAsposeVSDX.

The InputAdapter abstract class and its concrete implementation holds a representation of a sequence diagram from an external library, and its primary use is to read this information. The OutputAdapter abstract class and its concrete implementation is used to setup the external library's file output data structure, and it performs the actual output file creation process.

3.2. Concurrent Process Decomposition

This Java library does not involve any concurrent processes.

3.3. Data Decomposition

Both the input and output adapters contain and read from / write to data structures from external libraries. The details of these data structures depend on the external libraries used. The information used for creating the output file is given as attributes in the SDMtoFile class. Finally, the type information and extension string are given as attributes in the concrete extensions to the OutputType class.

3.4. State Model Decomposition

A state transition diagram is provided in the SRS, under the appendix (section 3). A supporting data flow diagram is provided in the SRS, under the appendix (section 2).

3.5. Use Case Model Decomposition

A use case diagram is provided in the SRS, under the appendix (section 1).

4 Dependency Descriptions

4.1. Intermodule Dependencies

The *SDMfileGenerator* package relies on the external Aspose.Diagram package for output, and the external UMLTranslator package for input, in order to create a file from the sequence diagram information given in a UMLSequenceDiagram object. As required in the SRS, section 3.1.3.

4.2. Interprocess Dependencies

Descriptions of all interprocess dependencies are given using UML notation in the appendix under “Architecture Class Diagram”, and details not specified there are given here. For a concrete instance of the OutputAdapter class, in its saveToFile() method, the information given in an OutputType descendant class is used when determining the file extension and which file format to use. This object is passed as a parameter to the saveToFile() method. For concrete instances of an OutputType class, static parameters in the Aspose.Diagram package’s SaveFileFormat class are used in the value of the TYPE attribute. For the InputBeulahWorks class, various methods in the UMLTranslator package are used to read the information in the contained UMLSequenceDiagram object. For the OutputAspose class, various methods in the Aspose.Diagram package are used to write to the contained and internally created Diagram object. This Diagram object is then used by Aspose.Diagram to create and format the output file. Many class methods may throw a library-specific Exception called SDMException when needed.

4.3. Data Dependencies

Data dependencies are given either in the appendix under “Architecture Class Diagram”, or in the previous sections. The most important inter-package dependencies include an attribute of the InputBeulahWorks class, which contains a UMLSequenceDiagram from the UMLTranslator package, supplied to it via a constructor. The OutputAspose class contains a Diagram object from the Aspose.Diagram package, generated internally when the initialize() method is called.

4.4. State Dependencies

The InputAdapter class handles the “Get SDM Model” transition, as shown in the SRS appendix (section 3), the SDMtoFile class handles the “Extract Sequence Diagram” state, and the OutputAdapter class handles the “Generate Visio File” state. See the “Setup Sequence Diagram” and the “Export File Sequence Diagram”, both located in the appendix, for more details.

4.5. Layer Dependencies

This library depends on both the Aspose.Diagram and UMLTranslator packages.

5 Interface Description

5.1. Module Interfaces

5.1.1. Interface to the SDMfileGenerator Package

The interface to the SDMfileGenerator package is provided by the SDMtoFile class. This class has a method and constructor parameter for taking in an OutputType object, which determines the output file's type and extension. It also has a method and constructor parameter for taking in an OutputAdapter object, which is used to interface with a file output library. This object is created by default as an OutputAspose object. Finally, its exportFile() method takes in an InputAdapter object, which is used to interface with a sequence diagram representation library. It may also optionally take in a PrintStream object used to log the export process.

5.1.2. Interface to the Beulah Works' UMLTranslator Package

The interface to the *UMLTranslator* package is provided by the InputBeulahWorks class. A UMLSequenceDiagram attribute object is the primary component of this class. This object is passed as a parameter of the constructor for this class. Section 6 gives details on the methods used by this class to interface with this object and its package.

5.1.3. Interface to the Aspose.Diagram Package

The interface to the *Aspose.Diagram* package is provided by the OutputAspose class. A Diagram attribute object is the primary component of this class. This object is created by this class when its initialize() method is run. Section 6 gives details on the methods used by this class to interface with this object and its package.

5.2. Process Interfaces

The method interfaces to each of the provided classes are described here.

SDMtoFile:

1. setOverwrite(Boolean) //Sets whether to overwrite an already existing file on file output.
2. setOutputFile(path, name) //Sets the output file directory and file name, minus extension.
3. setOutputType(OutputType) //Sets the output file's type using an OutputType object.
4. setOutputAdapter(OutputAdapter) //Sets the OutputAdapter object used for file saving.
5. exportFile(InputAdapter) //Exports the sequence diagram in the InputAdapter object.
6. exportFile(InputAdapter, PrintStream) //Same as above, with logging capabilities.

***OutputType*:**

1. getExtension() //Gets the file extension string of this class' concrete implementation.
2. getType() //Gets the file type of this class' concrete implementation.

OutputTypeAsposeVSDX:

Same as OutputType.

***InputAdapter*:**

1. getActorCount() //Gets the number of actor elements in the diagram.
2. getActorName(index) //Gets the name of the actor at the given index.
3. getClassBlockCount() //Gets the number of object elements in the diagram.
4. getClassBlockInstanceName(index) //Gets the name of the object at the given index.
5. getClassBlockClassName(index) //Gets the name of the object's class at the given index.
6. getActivationBlockCount() //Gets the number of process block elements in the diagram.
7. getLifelineCount() //Gets the number of connection between processes and elements.
8. getLifelineFromIndex(index) //Gets the starting element's index.
9. getLifelineToIndex(index) //Gets the ending element's index.
10. getLifelineActive(index) //Gets whether the connected object exists.
11. getMethodCount() //Gets the number of methods between multiple processes.
12. getMethodFromIndex(index) //Gets the starting element's index.
13. getMethodToIndex(index) //Gets the ending element's index.
14. getMethodText(index) //Gets the method name and other method text.
15. getConstraintCount() //Gets the number of constraint elements.
16. getConstraintText(index) //Gets the constraint's text, aka its constraint.
17. getLoopCount() //Gets the number of loop boxes.
18. getLoopText(index) //Gets the loop condition / constraint.
19. getAlternativeCount() //Gets the number of if-statement boxes.
20. getAlternativeText(index) //Gets the condition needed for the first section.
21. getAlternativeTextElse(index) //Gets the text for the else section.

InputBeulahWorks:

Same as InputAdapter.

***OutputAdapter*:**

1. initializeDiagram() //Create the internal data structures / object used for exporting files.
2. addActor(name) //Add an actor element to the sequence diagram with the given name.
3. addClassBlock(name, class) //Add an object element with the given name and class.
4. addActivationBlocks(count) //Add process block elements to the internal diagram.
5. addLifeline(fromIndex, toIndex, active) //Connect two added elements with a line.
6. addMethod(fromIndex, toIndex, text) //Add a method call message between processes.
7. addConstraint(text) //Add a constraint box with the given constraint text.
8. addLoop(text) //Add a loop box with the given condition text.
9. addAlternative(text, textElse) //Add an if-statement box with the given condition text.
10. finalizeDiagram() //Does the diagram element placement, with other finalization code.
11. saveToFile(path, name, type, overwrite) //Saves all the added elements to a diagram file.

OutputAspose:

Same as OutputAdapter.

SDMException:

Same as Exception. Used to determine when a library-specific exception is thrown.

6 Detailed Design

This section describes the detailed design of the classes and methods contained in the SDMfileGenerator package. See the "Master Class Diagram" and Sequence Diagrams located in the Appendix for more detailed design information.

6.1. Module Detailed Design

6.1.1. SDMtoFile

<u>Class Invariants</u>

No attributes are null.
filePath and fileName have valid path and name strings for the run-time operating system.
fileType and outputAdapter both contain concrete implementations.

<u>Attributes</u>

private boolean overwrite

This is set to true to allow overwriting a file with the same name as the one to be exported, if it exists.
Default: False

private String filePath

This represents the file path that the exported file will be saved to.
Default: .jar's start-in directory when run.

private String fileName

This represents the file name of the exported file, without the extension.
Default: "SequenceDiagram"

private OutputType fileType

This contains the object in which the extension (of the file) and the file type data structure are held.
Default: OutputTypeAsposeVSDX()

private OutputAdapter outputAdapter

This contains the object that creates the output data structure and exports a sequence diagram to a file.
Default: OutputAspose()

<u>Constructors</u>

public SDMtoFile()

Preconditions: None
Postconditions: Creates a SDMtoFile instance with the default attributes.

public SDMtoFile(String path, String name)

Preconditions: None
Postconditions: Creates a SDMtoFile instance with the given file path and name.

public SDMtoFile(String path, String name, OutputType type)
Preconditions: None
Postconditions: Creates a SDMtoFile instance as above, while also setting the file output type object.

public SDMtoFile(String path, String name, OutputType type, OutputAdapter adapter)
Preconditions: None
Postconditions: Creates a SDMtoFile instance as above, while also setting the output adapter object.

Methods
public void setOverwrite(boolean overwriteFile)
Preconditions: None
Postconditions: The overwrite attribute is set to the value contained in overwriteFile.

public void setOutputFile(String path, String name)
Preconditions: Parameter must not be null or improperly formatted.
Postconditions: The filePath and fileName attributes are set to the corresponding values of the parameters.

public void setOutputType(OutputType type)
Preconditions: The parameter must not be null, and must be a concrete object.
Postconditions: The fileType attribute is set to that stored in the type parameter.

public void setOutputAdapter(OutputAdapter adapter)
Preconditions: The parameter must not be null, and must be a concrete object.
Postconditions: The outputAdapter attribute is set to that stored in the adapter parameter.

public void exportFile(InputAdapter diagram)
Preconditions: The diagram parameter must not be null, and must be a concrete object.
Postconditions: A file containing the parameter's internal sequence diagram will be saved using the information found in the SDMtoFile object's attributes.
Throws: SDMException if the file export process does not successfully complete.

public void exportFile(InputAdapter diagram, PrintStream log)
Preconditions: The diagram parameter must not be null, and must be a concrete object.
Postconditions: Same as above, with the file export status printed to the log parameter while this runs.
Throws: Same as above.

6.1.2. OutputType<TypeInfo>

Methods
public abstract String getExtension()
Preconditions: None
Postconditions: Returns a string containing the extension part of a file name.

public abstract TypeInfo getType()
Preconditions: None
Postconditions: Returns a TypeInfo object using Java generics, used by the output adapter to format the file.

6.1.3. OutputTypeAsposeVSDX

Inheritance
This class extends the class OutputType<Integer>.

Attributes
private final String EXTENSION
This represents the file name's extension.
Default / Permanent Value: ".vsdx"

private final int TYPE
This represents the type that OutputAspose uses to determine the output file's format / type.
Default / Permanent Value: aspose.diagram.SaveFileFormat.VSDX

Methods
public String getExtension()
Preconditions: None
Postconditions: Same as OutputType, Returns EXTENSION

public Integer getType()
Preconditions: None
Postconditions: Same as OutputType, Returns TYPE

6.1.4. InputAdapter

Methods
public int getActorCount()
Preconditions: None
Postconditions: Returns the number of actor elements in the sequence diagram.

public String getActorName(int index)
Preconditions: index must refer to a valid actor in the sequence diagram.
Postconditions: Returns the given actor element's name.

public int getClassBlockCount()
Preconditions: None
Postconditions: Returns the number of object elements in the sequence diagram.

public String getClassBlockInstanceName(int index)
Preconditions: index must refer to a valid object in the sequence diagram.
Postconditions: Returns the given object element's name.

public String getClassBlockClassName(int index)
Preconditions: index must refer to a valid object in the sequence diagram.
Postconditions: Returns the given object element's class name.

public int getActivationBlockCount()
Preconditions: None
Postconditions: Returns the number of process elements in the sequence diagram.

public int getLifelineCount()
Preconditions: None
Postconditions: Returns the number of connections between elements in the sequence diagram.

public int getLifelineFromIndex(int index)
Preconditions: index must refer to a valid connection in the sequence diagram.
Postconditions: Returns the index of the connection's starting element.

public int getLifelineToIndex(int index)
Preconditions: index must refer to a valid connection in the sequence diagram.
Postconditions: Returns the index of the connection's ending element.

public boolean getLifelineActive(int index)
Preconditions: index must refer to a valid connection in the sequence diagram.
Postconditions: Returns the whether the connection refers to an object that currently exists.

public int getMethodCount()
Preconditions: None
Postconditions: Returns the number of method connections between elements in the sequence diagram.

public int getMethodFromIndex(int index)
Preconditions: index must refer to a valid method in the sequence diagram.
Postconditions: Returns the index of the method's starting element.

public int getMethodToIndex(int index)
Preconditions: index must refer to a valid method in the sequence diagram.
Postconditions: Returns the index of the method's ending element.

public String getMethodText(int index)
Preconditions: index must refer to a valid method in the sequence diagram.
Postconditions: Returns the method's name and other related method text.

public int getConstraintCount()
Preconditions: None
Postconditions: Returns the number of constraint elements in the sequence diagram.

public String getConstraintText(int index)
Preconditions: index must refer to a valid constraint in the sequence diagram.
Postconditions: Returns the constraint's inner text.

public int getLoopCount()
Preconditions: None
Postconditions: Returns the number of loop block elements in the sequence diagram.

public String getLoopText(int index)
Preconditions: index must refer to a valid loop block in the sequence diagram.
Postconditions: Returns the looping condition's inner text.

public int getAlternativeCount()
Preconditions: None
Postconditions: Returns the number of alternative (if-else) block elements in the sequence diagram.

public String getAlternativeText(int index)
Preconditions: index must refer to a valid alternative (if-else) statement block in the sequence diagram.
Postconditions: Returns the if statement's inner text.

public String getAlternativeTextElse(int index)
Preconditions: index must refer to a valid alternative (if-else) statement block in the sequence diagram.
Postconditions: Returns the else statement's inner text.

6.1.5. InputBeulahWorks

Class Invariants
The diagram attribute is not null.

Inheritance
This class implements the class InputAdapter.

Attributes
private UMLSequenceDiagram diagram
This contains the data structures to read from, as given in the Beulah Work's SDM package.
Default: Given via constructor.

Constructors
private InputBeulahWorks()
Preconditions: None
Postconditions: None

public InputBeulahWorks(UMLSequenceDiagram diagram)
Preconditions: The diagram parameter must not be null.
Postconditions: The diagram attribute is set to the contents of the diagram parameter.

Methods
Same as InputAdapter.

6.1.6. OutputAdapter

Methods
public String initializeDiagram()
Preconditions: None
Postconditions: The output library's diagram object is created or reset. Logging information is returned.

public String addActor(String actorName)
Preconditions: None
Postconditions: An actor diagram element with the given name is added to the diagram. Logging information is returned.

public String addClassBlock(String instanceName, String className)
Preconditions: None
Postconditions: An object diagram element with the given name and class text is added to the diagram. Logging information is returned.

public String addActivationBlocks(int count)
Preconditions: None
Postconditions: The given number, count, of process block diagram elements are added to the diagram. Logging information is returned.

public String addLifeline(int fromIndex, int toIndex, boolean active)
Preconditions: fromIndex and toIndex must refer to valid, previously added elements.
Postconditions: A connection starting at the element referred to by fromIndex, and ending at that referred to by toIndex, is added to the diagram. With a dashed line when active is false, and a solid line otherwise. Logging information is returned.

public String addMethod(int fromIndex, int toIndex, String text)
Preconditions: fromIndex and toIndex must refer to valid, previously added elements.
Postconditions: A method starting at the element referred to by fromIndex, and ending at that referred to by toIndex, with name for the method name, is added to the diagram. Logging information is returned.

public String addConstraint(String text)
Preconditions: None
Postconditions: A constraint diagram element with the given internal text is added to the diagram. Logging information is returned.

public String addLoop(String text)
Preconditions: None
Postconditions: A loop block diagram element with the given internal text for the looping condition is added to the diagram. Logging information is returned.

public String addAlternative(String text, String textElse)
Preconditions: None
Postconditions: An alternative (if-else) block diagram element, with the given internal text for the if condition, and textElse for the else text, is added to the diagram. Logging information is returned.

public String finalizeDiagram()
Preconditions: None
Postconditions: The output library's diagram elements are positioned. Logging information is returned.

public String saveToFile(String path, String name, OutputType type, boolean overwrite)
Preconditions: All parameters must be valid, as given in SDMtoFile, and not null.
Postconditions: A file containing all the information previously added to the diagram attribute is created. Logging information is returned.

6.1.7. OutputAspose

Class Invariants

The diagram attribute is not null.

Inheritance

This class implements the class OutputAdapter.

<u>Attributes</u>
private Diagram diagram
This stores the aspose.diagram data structure used to generate the file.
Default: An Aspose Diagram object with master stencils added and no sequence diagram information.

<u>Constructors</u>
public OutputAspose()
Preconditions: None
Postconditions: Creates an OutputAspose instance and calls initialize() on it.

<u>Methods</u>
Same as OutputAdapter.

6.1.8. SDMException

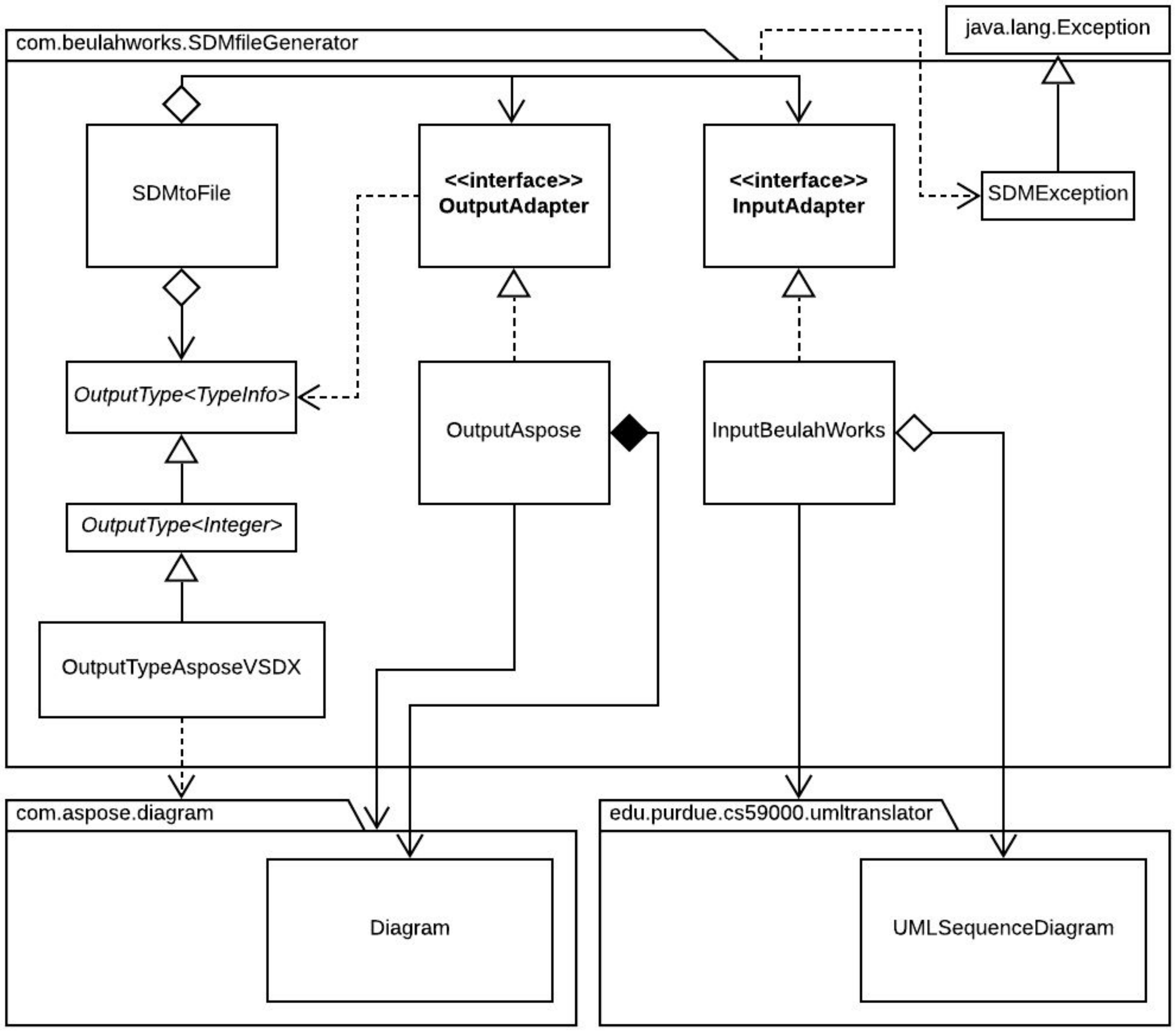
<u>Inheritance</u>
This class extends the class Exception.

6.2. Data Detailed Design

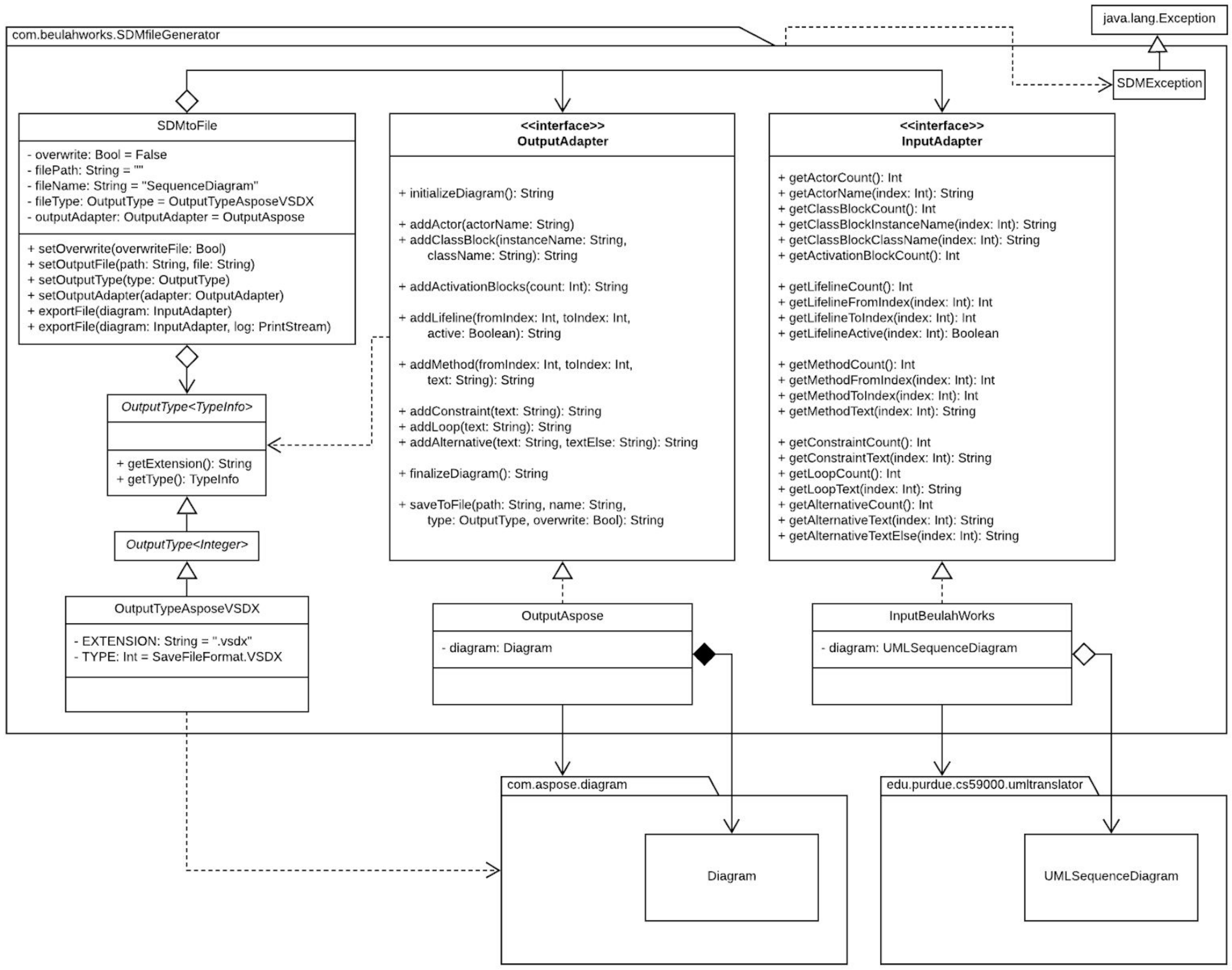
There are no data structures besides those that are part of the classes in Section 6.1.

Appendix

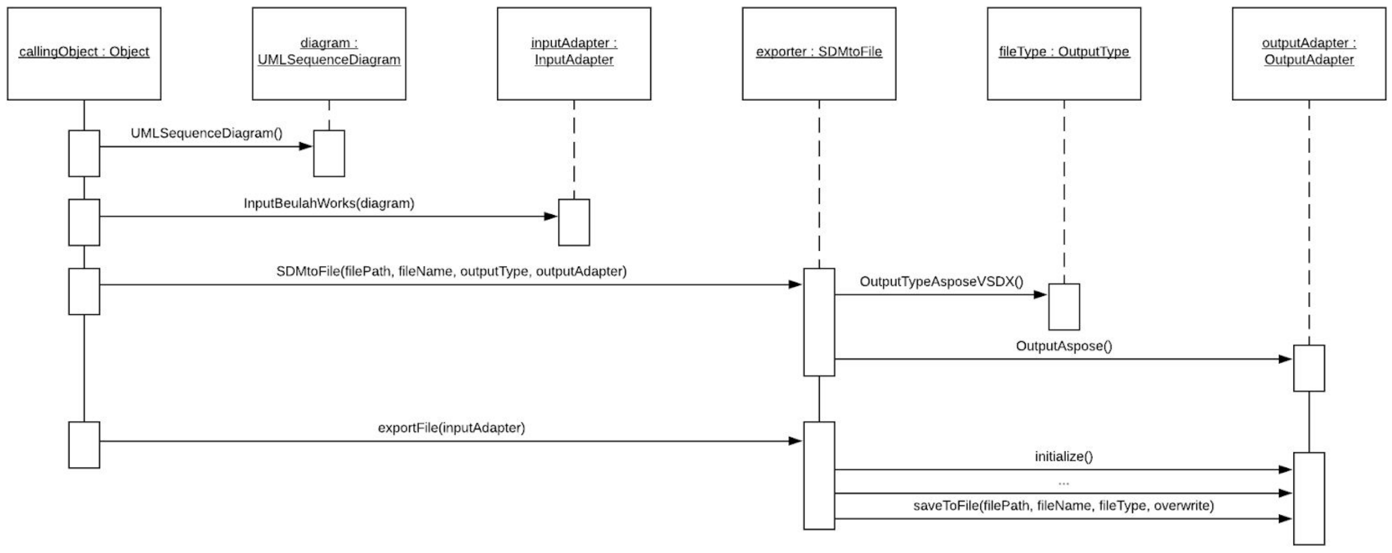
1. Architecture Class Diagram



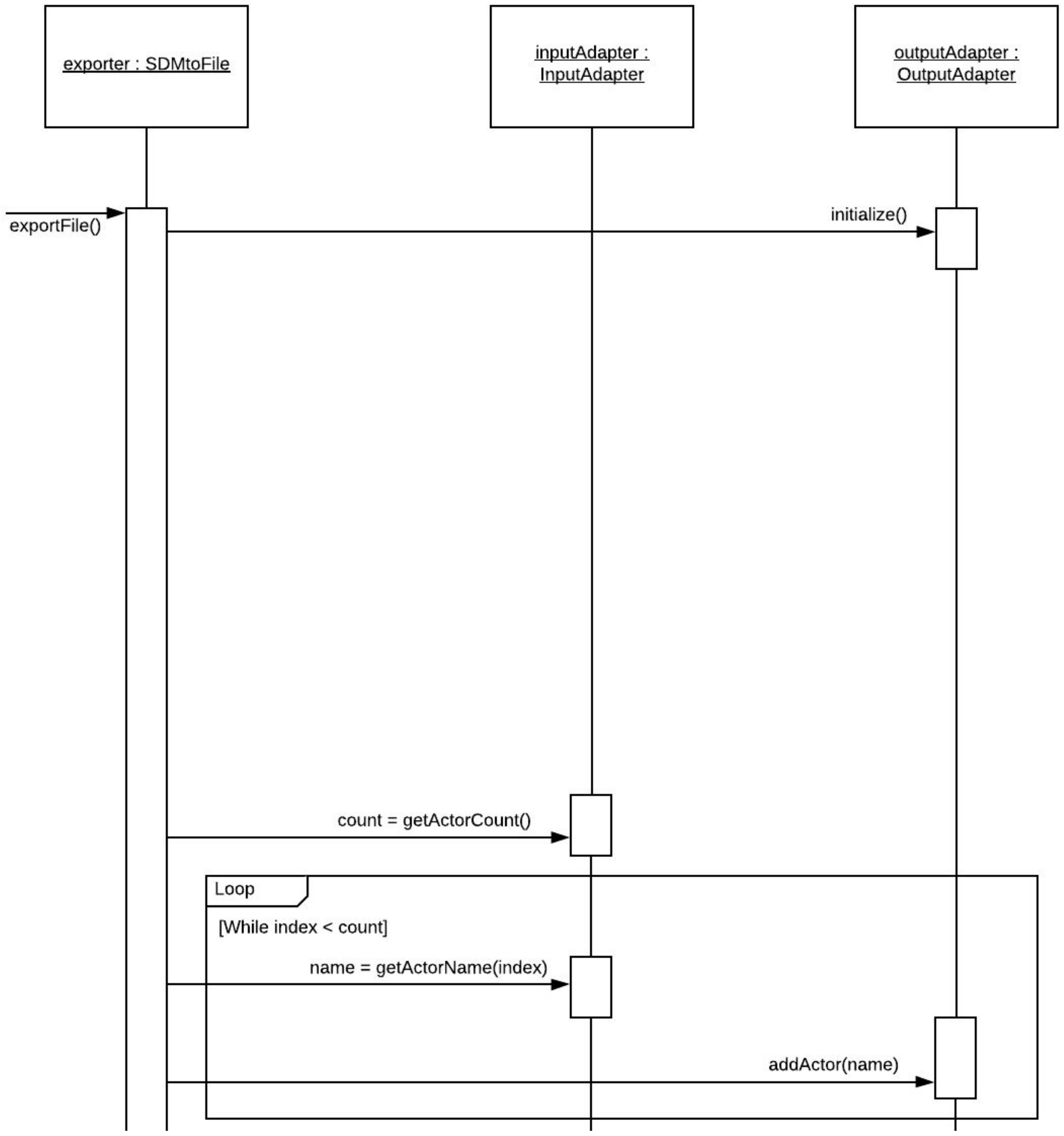
2. Master Class Diagram

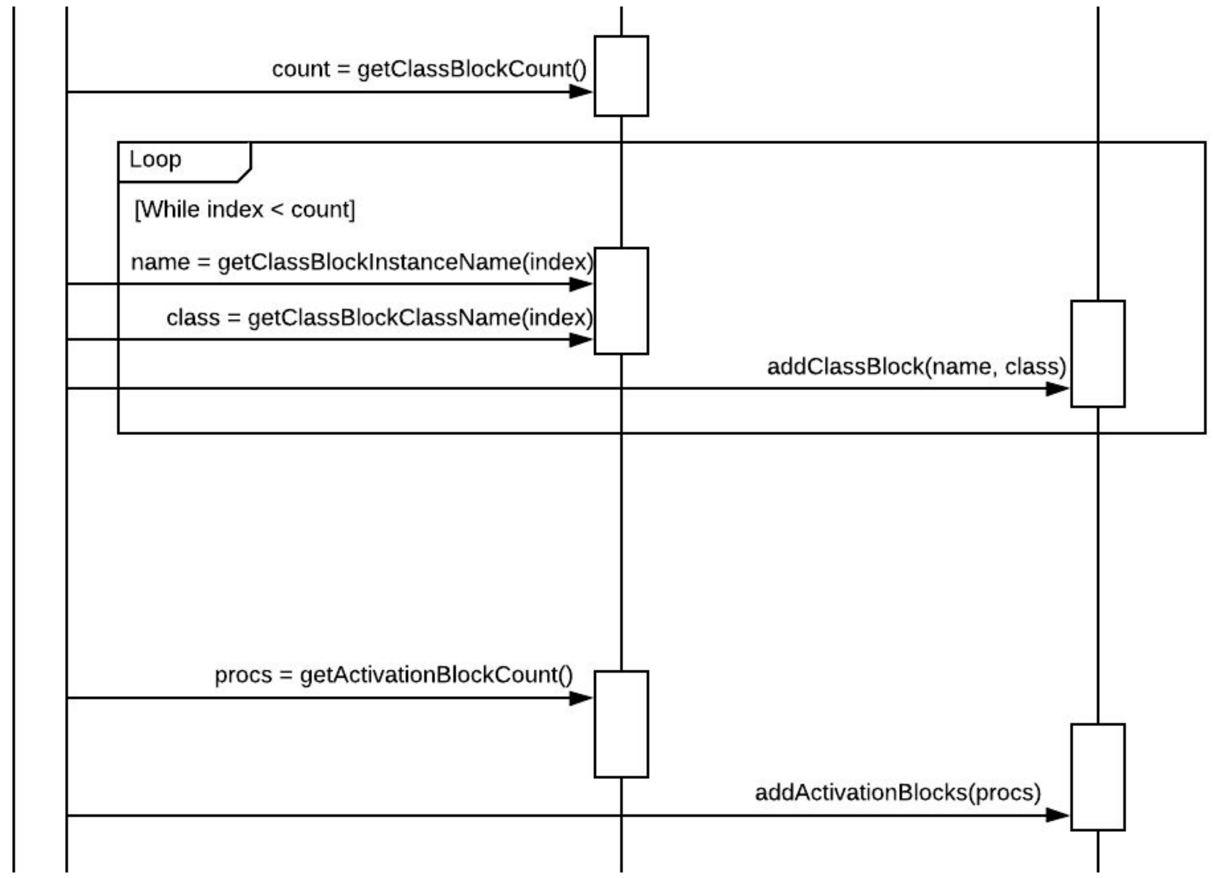

java.lang.Exception
com.beulahworks.SDMfileGenerator
SDMException
SDMtoFile
- overwrite: Bool = False
- filePath: String = ""
- fileName: String = "SequenceDiagram"
- fileType: OutputType = OutputTypeAsposeVSDX
- outputAdapter: OutputAdapter = OutputAspose
+ setOverwrite(overwriteFile: Bool)
+ setOutputFile(path: String, file: String)
+ setOutputType(type: OutputType)
+ setOutputAdapter(adapter: OutputAdapter)
+ exportFile(diagram: InputAdapter)
+ exportFile(diagram: InputAdapter, log: PrintStream)
<<interface>> OutputAdapter
+ initializeDiagram(): String
+ addActor(actorName: String)
+ addClassBlock(instanceName: String, className: String): String
+ addActivationBlocks(count: Int): String
+ addLifeline(fromIndex: Int, toIndex: Int, active: Boolean): String
+ addMethod(fromIndex: Int, toIndex: Int, text: String): String
+ addConstraint(text: String): String
+ addLoop(text: String): String
+ addAlternative(text: String, textElse: String): String
+ finalizeDiagram(): String
+ saveToFile(path: String, name: String, type: OutputType, overwrite: Bool): String
<<interface>> InputAdapter
+ getActorCount(): Int
+ getActorName(index: Int): String
+ getClassBlockCount(): Int
+ getClassBlockInstanceName(index: Int): String
+ getClassBlockClassName(index: Int): String
+ getActivationBlockCount(): Int
+ getLifelineCount(): Int
+ getLifelineFromIndex(index: Int): Int
+ getLifelineToIndex(index: Int): Int
+ getLifelineActive(index: Int): Boolean
+ getMethodCount(): Int
+ getMethodFromIndex(index: Int): Int
+ getMethodToIndex(index: Int): Int
+ getMethodText(index: Int): String
+ getConstraintCount(): Int
+ getConstraintText(index: Int): String
+ getLoopCount(): Int
+ getLoopText(index: Int): String
+ getAlternativeCount(): Int
+ getAlternativeText(index: Int): String
+ getAlternativeTextElse(index: Int): String
OutputType<TypeInfo>
+ getExtension(): String
+ getType(): TypeInfo
OutputType<Integer>
OutputTypeAsposeVSDX
- EXTENSION: String = ".vsdx"
- TYPE: Int = SaveFileFormat.VSDX
OutputAspose
- diagram: Diagram
InputBeulahWorks
- diagram: UMLSequenceDiagram
com.aspose.diagram
Diagram
edu.purdue.cs59000.umltranslator
UMLSequenceDiagram

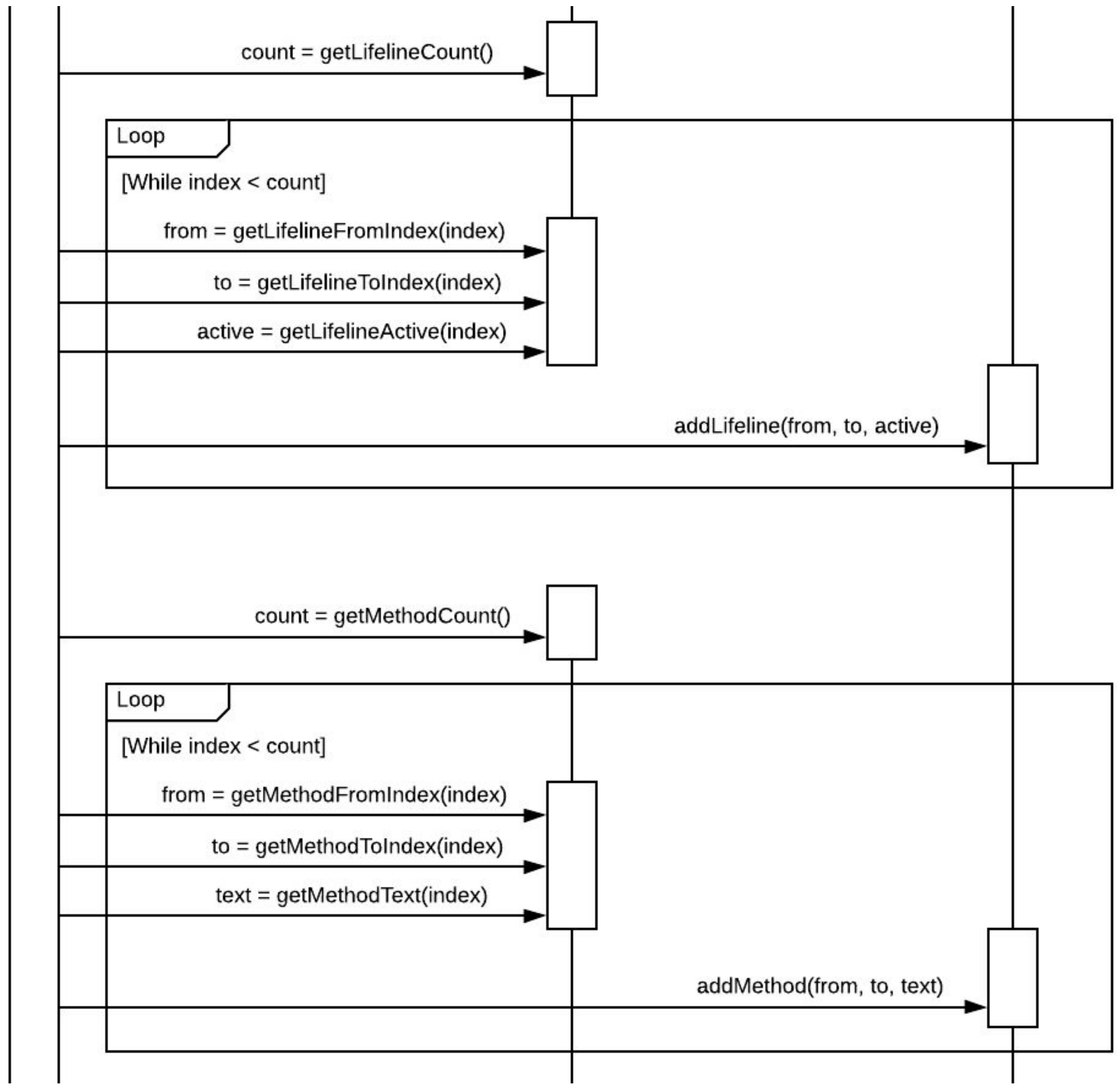
3. Setup Sequence Diagram

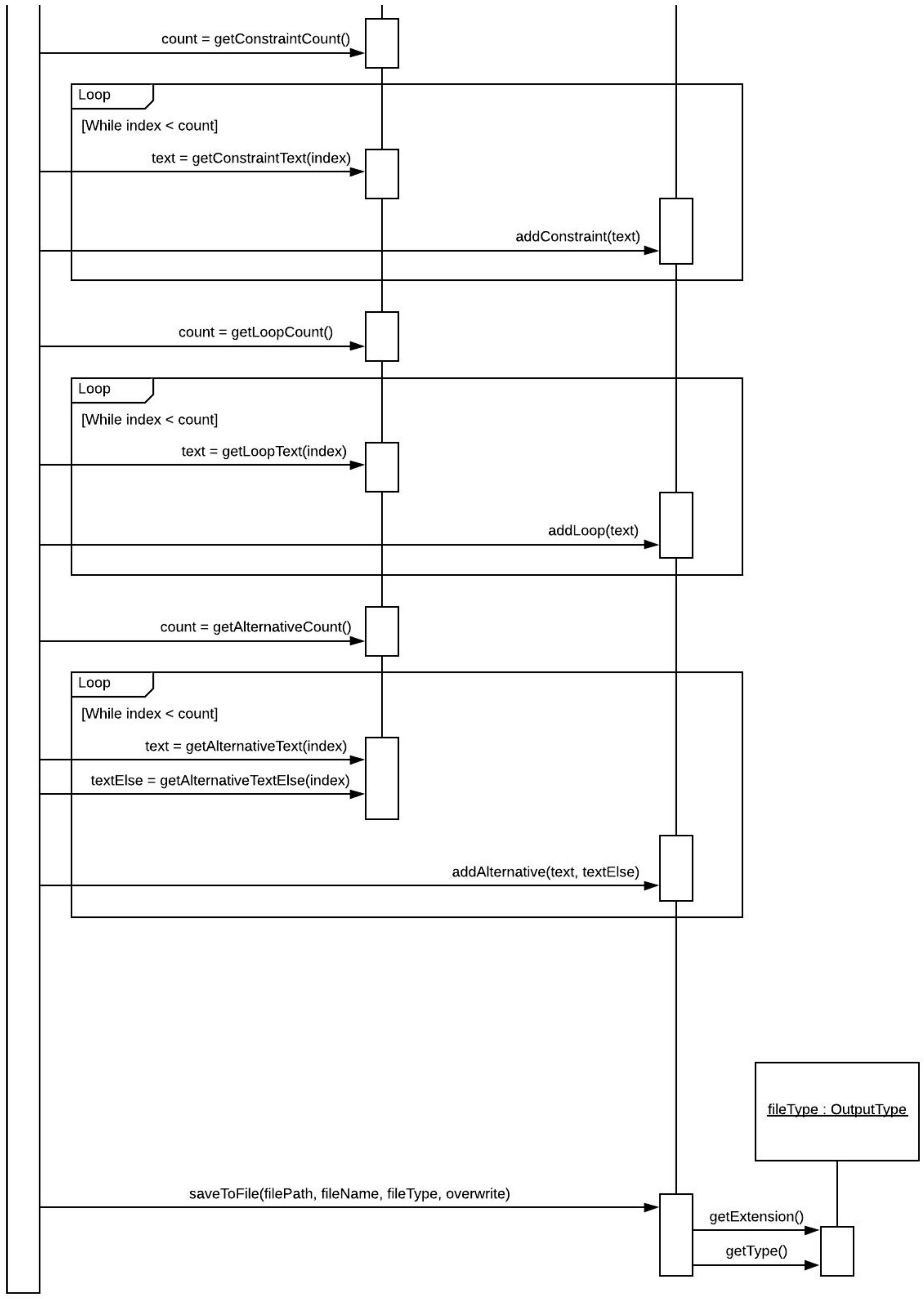

callingObject : Object
diagram : UMLSequenceDiagram
inputAdapter : InputAdapter
exporter : SDMtoFile
fileType : OutputType
outputAdapter : OutputAdapter
UMLSequenceDiagram()
InputBeulahWorks(diagram)
SDMtoFile(filePath, fileName, outputType, outputAdapter)
OutputTypeAsposeVSDX()
OutputAspose()
exportFile(inputAdapter)
initialize()
...
saveToFile(filePath, fileName, fileType, overwrite)

4. Export File Sequence Diagram




count = getClassBlockCount()
Loop
[While index < count]
name = getClassBlockInstanceName(index)
class = getClassBlockClassName(index)
addClassBlock(name, class)
procs = getActivationBlockCount()
addActivationBlocks(procs)


count = getLifelineCount()
Loop
[While index < count]
from = getLifelineFromIndex(index)
to = getLifelineToIndex(index)
active = getLifelineActive(index)
addLifeline(from, to, active)
count = getMethodCount()
Loop
[While index < count]
from = getMethodFromIndex(index)
to = getMethodToIndex(index)
text = getMethodText(index)
addMethod(from, to, text)


count = getConstraintCount()
Loop
[While index < count]
text = getConstraintText(index)
addConstraint(text)
count = getLoopCount()
Loop
[While index < count]
text = getLoopText(index)
addLoop(text)
count = getAlternativeCount()
Loop
[While index < count]
text = getAlternativeText(index)
textElse = getAlternativeTextElse(index)
addAlternative(text, textElse)
fileType : OutputType
saveToFile(filePath, fileName, fileType, overwrite)
getExtension()
getType()