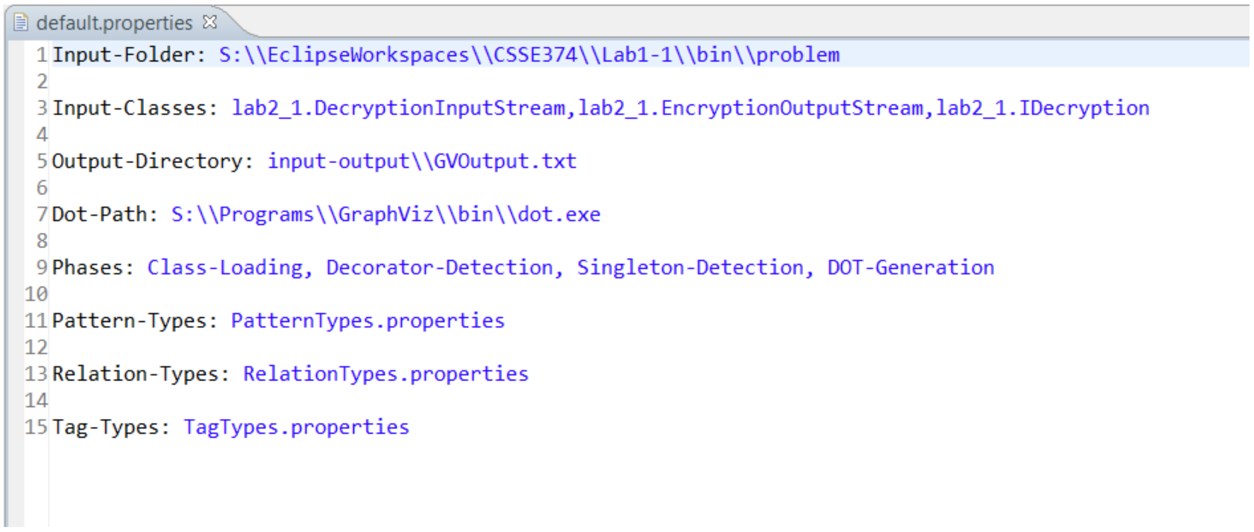
Supporting Documentation - UML Generator by Eleven

This document contains information regarding the settings that users can configure in the .properties file. Included below is an image of an example properties file.



```
Input-Folder: S:\\EclipseWorkspaces\\CSSE374\\Lab1-1\\bin\\problem

Input-Classes: lab2_1.DecryptionInputStream,lab2_1.EncryptionOutputStream,lab2_1.IDecryption

Output-Directory: input-output\\GVOutput.txt

Dot-Path: S:\\Programs\\GraphViz\\bin\\dot.exe

Phases: Class-Loading, Decorator-Detection, Singleton-Detection, DOT-Generation

Pattern-Types: PatternTypes.properties

Relation-Types: RelationTypes.properties

Tag-Types: TagTypes.properties
```

Parsing Classes

There are two properties that the user can use to specify which Java classes will be parsed. The first is the “Input-Folder” property. Here, the user can specify the file path of a directory that contains Java .class files. Each .class file in the specified directory will be parsed and included in the UML diagram. The second way that users can specify classes is by using the “Input-Classes” property. Here, the user can specify fully qualified java classes (that are included in the project’s build path). This property is useful for parsing classes that might be hard to find in a directory, such as Java API classes (i.e. java.awt.Component, java.util.ArrayList).

Output

The user can specify a directory to place the special text output that is generated by our tool. The “Output-Directory” requires a valid file path that ends with a .txt file. If the specified file does not exist, a new one will be created at that location. This text file is used to hold the special text output that GraphViz requires in order to generate a UML diagram. It is very unlikely that the user will need to handle this output directly.

Dot Path

Our tool depends on GraphViz, specifically the dot.exe executable, to generate the UML diagram. The user must specify the location of this executable on their computer. The “Dot-Path” property requires a valid file path.

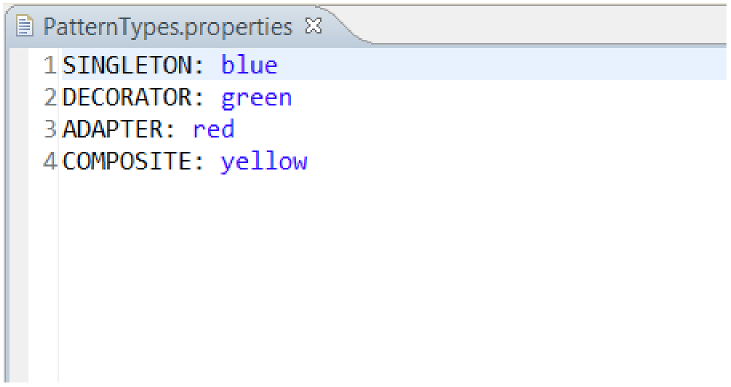
Phases

There are several phases that the user can configure. The phases are executed in the order that the user specifies. All currently implemented phases are listed below:

- Class-Loading: This phase uses ASM to parse each of the previously specified Java classes and build an intermediate model object. **This phase must be included as the very first phase** in order for our tool to generate a UML diagram.
- DOT-Generation: This phase uses the dot executable to convert the special text output into a proper UML diagram. This phase must be included in order to generate a UML diagram. Additionally, **this should be the very last phase** that is executed (as any phases included after it will be executed, but their changes will not be present in the UML diagram).
- Singleton-Detection: This phase runs singleton pattern detection on the intermediate model. Any singleton classes will be labeled with the <<Singleton>> stereotype and colored blue on the UML diagram.
- Decorator-Detection: This phase runs decorator pattern detection on the intermediate model. Any classes involved in a decorator pattern will be labeled with the appropriate stereotype and colored green on the UML diagram. Stereotypes include <<Decorator>> and <<Component>>. The association relationship between decorator and component classes will also be labeled with <<decorates>>.
- Adapter-Detection: This phase runs adapter pattern detection on the intermediate model. Any classes involved in an adapter pattern will be labeled with the appropriate stereotype and colored red on the UML diagram. Stereotypes include <<Adapter>>, <<Adaptee>>, and <<Target>>. The association relationship between the adapter and the adaptee will also be labeled with <<adapts>>.
- Composite-Detection: This phase runs composite pattern detection on the intermediate model. Any classes involved in a composite pattern will be labeled with the appropriate stereotype and colored yellow on the UML diagram. Stereotypes include <<Component>>, <<Composite>>, and <<Leaf>>.

Pattern Types

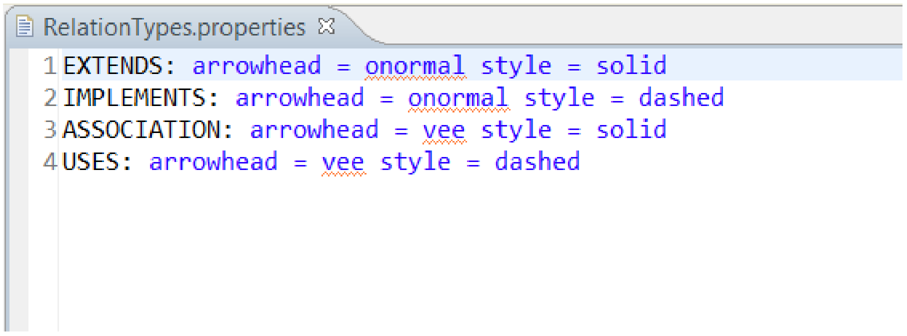
Our tool allows the user to configure the pattern types that are supported during detection and UML generation. The “Pattern-Types” property requires a file path to a valid .properties file. This properties file must include information regarding the supported pattern types and their colors.



```
SINGLETON: blue
DECORATOR: green
ADAPTER: red
COMPOSITE: yellow
```

Relation Types

Our tool allows the user to configure the relation types that are supported during detection and UML generation. The “Relation-Types” property requires a file path to a valid .properties file. This properties file must include information regarding the supported pattern types and details about their arrow types.


RelationTypes.properties

```
EXTENDS: arrowhead = onormal style = solid
IMPLEMENTS: arrowhead = onormal style = dashed
ASSOCIATION: arrowhead = vee style = solid
USES: arrowhead = vee style = dashed
```

Pattern-Specific Settings

Our tool supports the addition of pattern-specific settings. These settings are typically requirements that influence the logic that determines which classes qualify for specific patterns. Each pattern-specific setting can be included in the properties file as a standalone property. All currently implemented settings are listed below:

- Singleton-RequireGetInstance: This property requires a Boolean value (“true” or false”), and determines whether a class must have a “getInstance” method to qualify as a Singleton class. If this property is omitted from the properties file, it is internally set to false by default.