# Bluprint Architecture

## Overview

Bluprint adopts a simple three-layer architecture summarised in the following diagram. It illustrates the packages within the application, their dependency on one another and on third-party packages such the log4j. In keeping with a layered architecture, each layer is dependent upon the layer below, but not vice-versa.

This diagram and those shown subsequently are all taken from the UML MagicDraw model used to generate the Bluprint application.

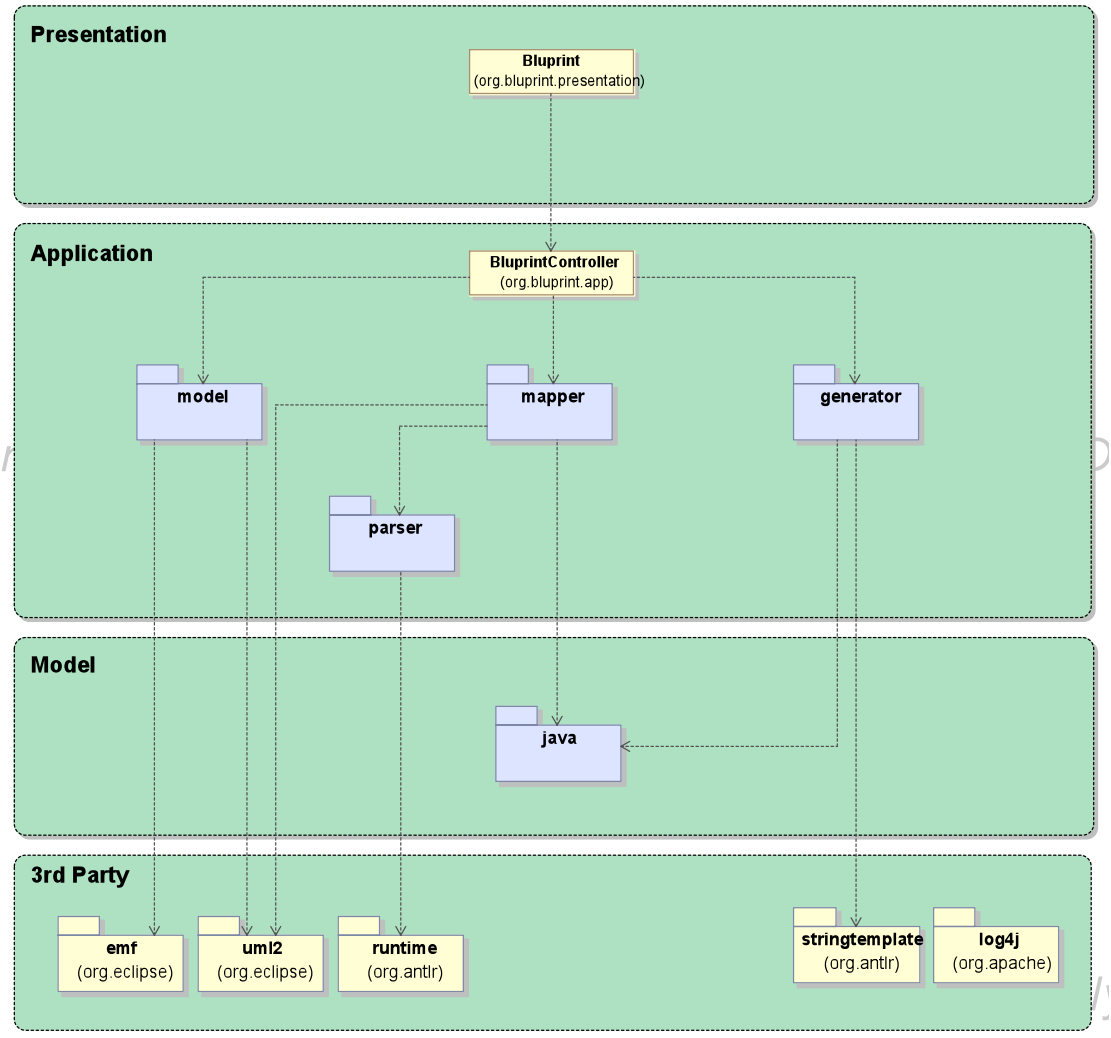


Figure 1 - Bluprint Architecture

The classes and packages within each layer, by convention, conform to a different broad set of responsibilities. In this architecture these are as follows:

- **Presentation**. This layer is responsible for supporting the presentation of the application, or user-interface, to the end user. The user-interface in

Bluprint is a simple command line interface that is implemented by the single class `Bluprint` in the `org.bluprint.presentation` package. Although the Presentation layer only contains a single class, it is possible for this to be replaced by a more sophisticated interface such as one implemented in Swing.

- **Application**. The Application layer is responsible for implementing the logic of the application in response to user requests forwarded from the Presentation Layer. It has a single entry point implemented in the class `BluprintController` within the `org.bluprint.app` package.
- **Model**. This layer contains the classes that support the underlying abstractions manipulated by the application. The package `org.bluprint.model.java` contains a set of classes that model java programs.

## Presentation Layer

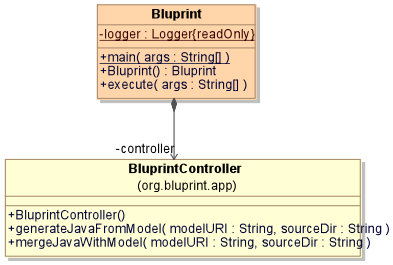


Figure 2 - org.bluprint.presentation

The Presentation layer has a single class called `Bluprint` that contains the entry point to the application via the `main()` function. This invokes the `execute()` function that parses the command line parameters and calls either the `generateJavaFromModel()` method or the `mergeJavaWithModel()` method within the `BluprintController` class, in the layer below, depending upon the number of parameters identified in the command line.

## Application Layer

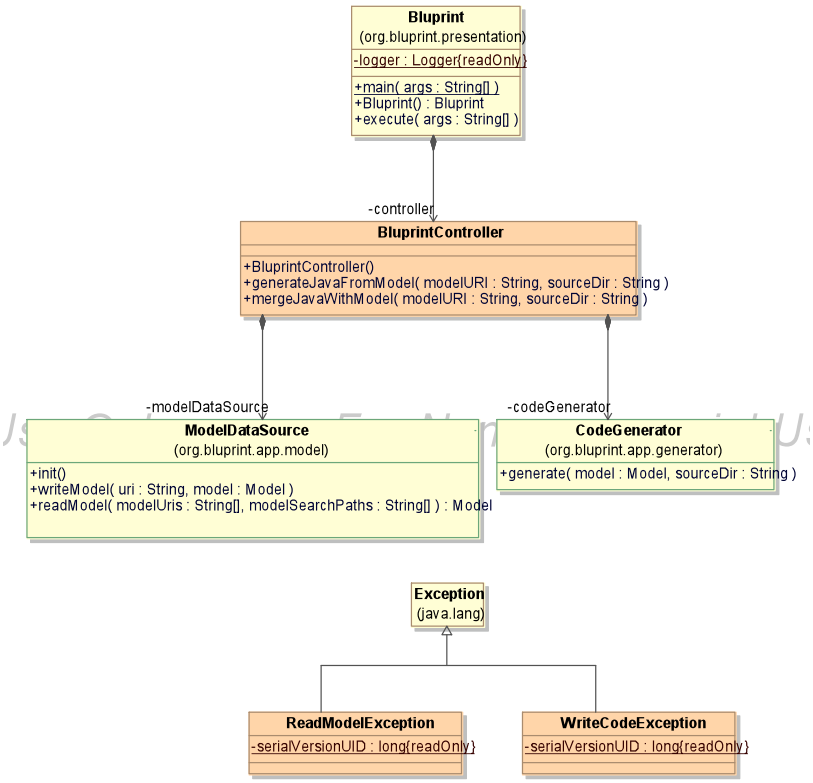


Figure 3 - org.bluprint.app

The `BluprintController` class exposes two methods for use by the Presentation layer, one for generating Java source code from an XMI UML model and the other for merging pre-existing Java source code with source generated from an XMI UML model. These are `generateJavaFromModel()` and `mergeJavaWithModel()` respectively.

### generateJavaFromModel()

The `generateJavaFromModel()` method performs the following logical steps:

1. Read, and parse, an XMI UML model from the specified file creating an in-memory representation of the model.
2. Invoke the code generator using the previously created in-memory representation of the model to create the Java source code in the specified output directory.

### mergeJavaWithModel()

In the `mergeJavaWithModel()` method the following logical steps are performed:

1. Read, and parse, an XMI UML model from the specified file creating an in-memory representation of the model.
2. Read, and parse, the Java source from the specified directory and sub-directories creating another in-memory representation of the code.
3. Merge both in-memory representations with one another.

4. Invoke the code generator using the merged in-memory representation to create the Java source code in the specified output directory.

## org.bluprint.app.model

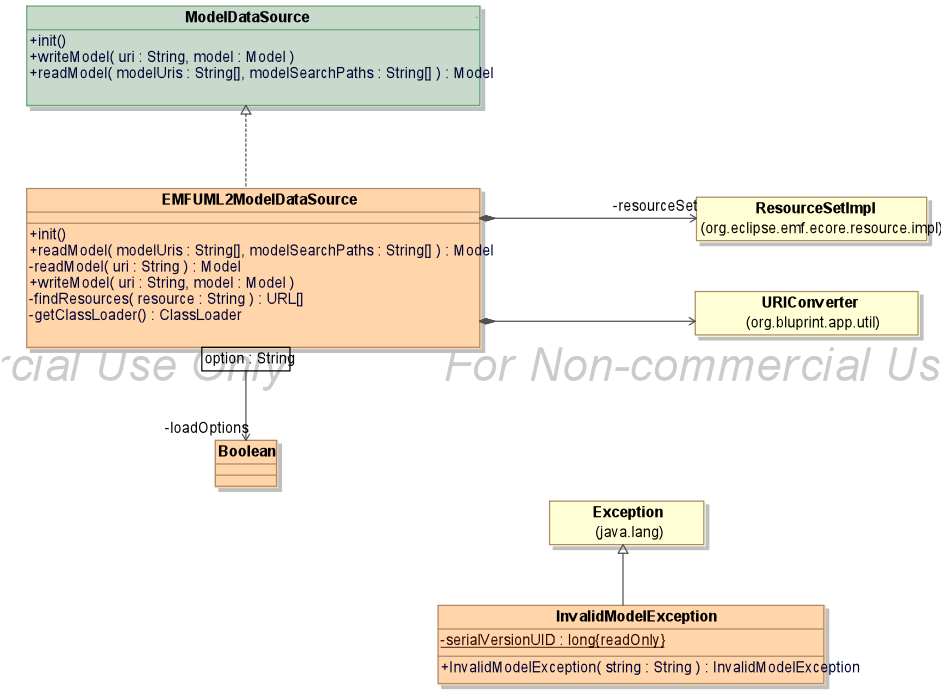




Figure 4 - org.bluprint.app.model

The `org.bluprint.app.model` package defines an interface for accessing and abstracting an externally maintained object model such an XMI model accessed via EMF UML2. Within the `EMFUML2ModelDataSource` class the `readModel()` and `writeModel()` methods implement the `ModelDataSource` interface to read and write `org.eclipse.uml2` Models.

## Modelling Third-party Classes and Libraries

As an aside third party classes, data types and APIs are maintained in the model, so for instance the classes in the `org.eclipe.uml2` package are modelled. The diagram below shows the interfaces defined in this package.

We model these classes so that we can reference them as parameters in method definitions or include them in our model diagrams.

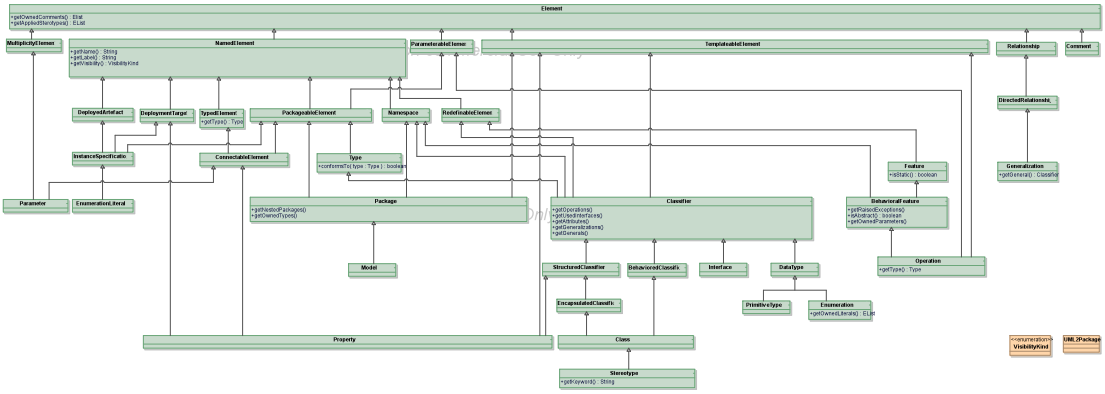

Figure 5 - org.eclipse.uml2

Bluprint recognises a custom stereotype called `<<nogen>>` which can be applied to classes, interfaces and packages within this model. The stereotype is used to indicate to the code generator within Bluprint that no code should be generated for that class, interface or package.

## org.bluprint.app.generator

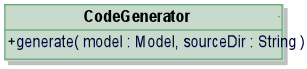


Figure 6 - org.bluprint.app.generator

This package defines a simple interface for the code generator with the intent that multiple different implementations could be provided. In Bluprint a single Java code generator is implemented, but it is the ambition of the project that other alternative generators could be created such as for C#.

## org.bluprint.app.generator.java

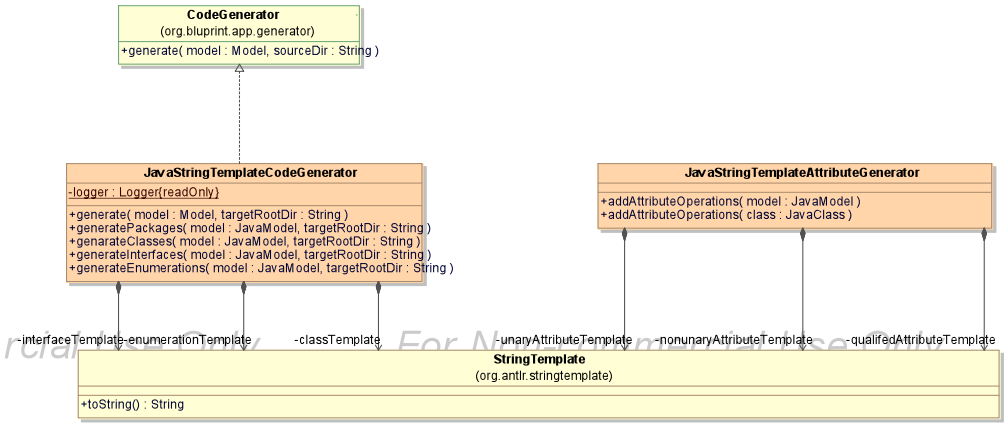

Figure 7 - org.bluprint.app.generator.java

This package contains the implementation of a Java code generator that takes an `org.bluprint.model.JavaModel`. It iterates over the model generating Java code using the `org.antlr.stringtemplate` third party library. This is a simple template library for writing formatted strings.

## org.bluprint.app.mapper

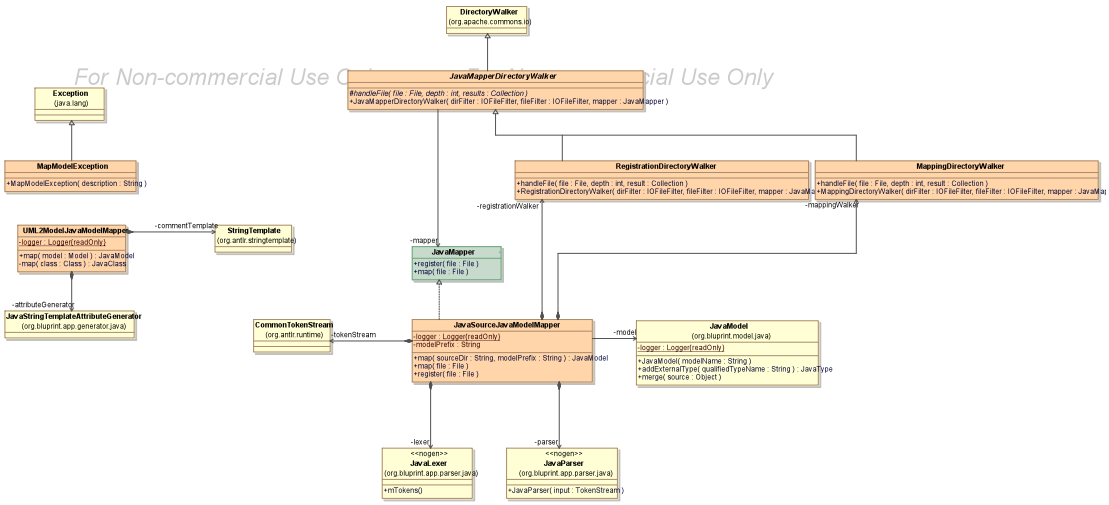


Figure 8 - org.bluprint.app.mapper

The `org.bluprint.app.mapper` package contains classes for mapping from various external model sources to a `org.bluprint.model.JavaModel`, Bluprint's abstract representation of a Java object model.

Two mapping classes are represented in this package, `UML2ModelJavaModelMapper` and `JavaSourceJavaModelMapper`. The former is used to map from an `org.eclipse.uml2.Model` to a `JavaModel` and the latter is used to map Java source code to a `JavaModel`.

The `UML2ModelJavaModelMapper` class is dependent upon the Antlr StringTemplate library as it is used for creating Java style comments from the comments contained in the EMF UML2 model as the Java code generator in this version of Bluprint is not very sophisticated. Comments in the `JavaModel` are maintained in a literal representation, usually to reflect the comments identified when parsing the Java source code. In a future version of Bluprint this should be refined so that comments in the `JavaModel` are stored in a Java source code independent format. This will be necessary when in a future release we abstract `JavaModel` to be a generic language independent model.

## org.bluprint.app.mapper.adapter

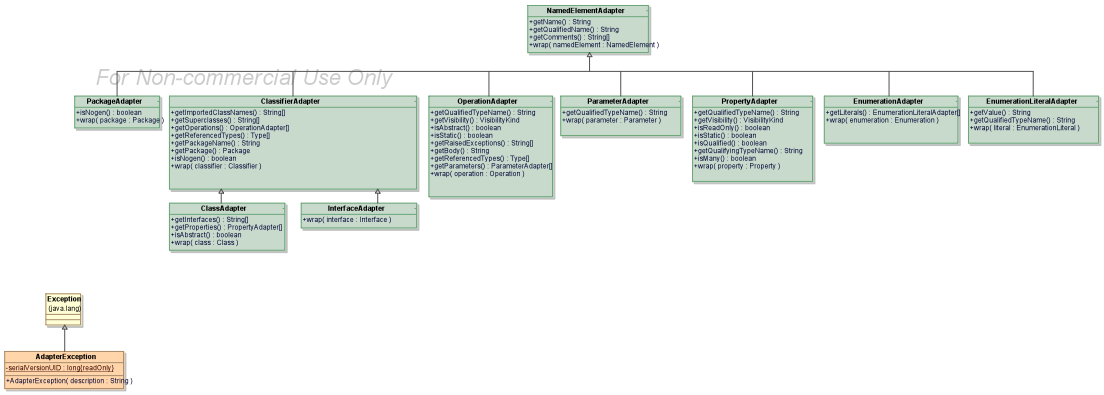


Figure 9 - org.bluprint.app.mapper.adapter

The interfaces in this package are used for abstracting access to an EMF UML2 object model. Interfaces are provided for the major constructs usually found in a model such as package, class, interface and operation. The intent of the design was to implement Bluprint against this adapter interface to simplify and isolate the code for iterating over and accessing elements of an EMF UML2 model. Notice that each interface implements a `wrap()` method which takes an underlying EMF UML2 class as a parameter. Each interface then provides a simple set of getter methods for accessing properties associated with the wrapped EMF UML2 class.

## org.bluprint.app.mapper.adapter.uml2

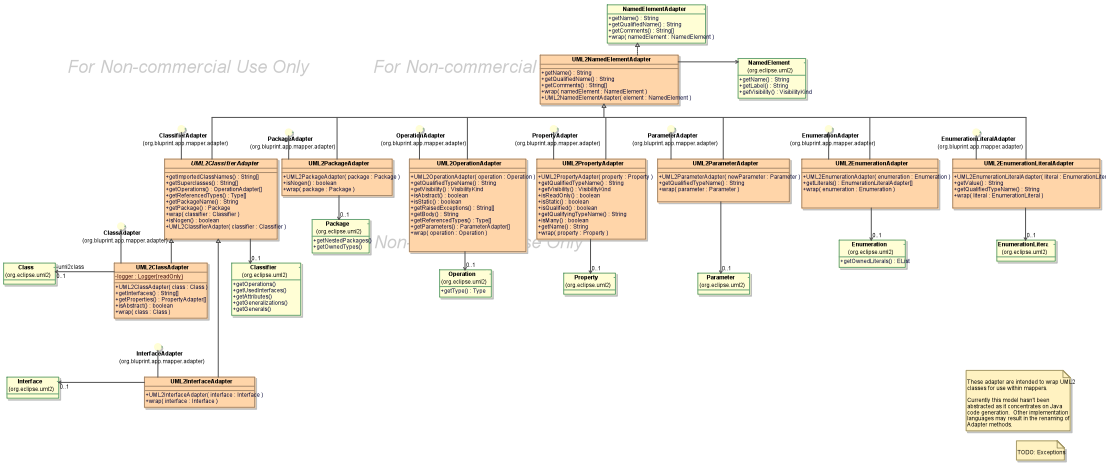

Figure 10 - org.bluprint.app.mapper.adapter.uml2

The above classes provide the concrete implementation for accessing an EMF UML2 model.

## org.bluprint.app.parser.java

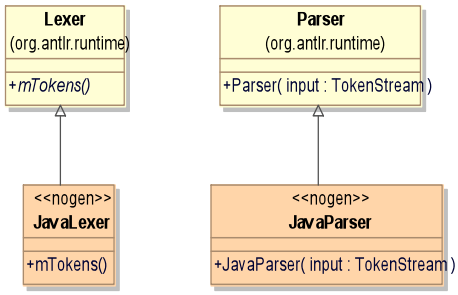


Figure 11 - org.bluprint.app.parser.java

Antlr is an open source parser generator and has been used to create a Java 1.5 parser for use by Bluprint. The output of the generator is represented in the model as shown above.

## org.bluprint.app.util

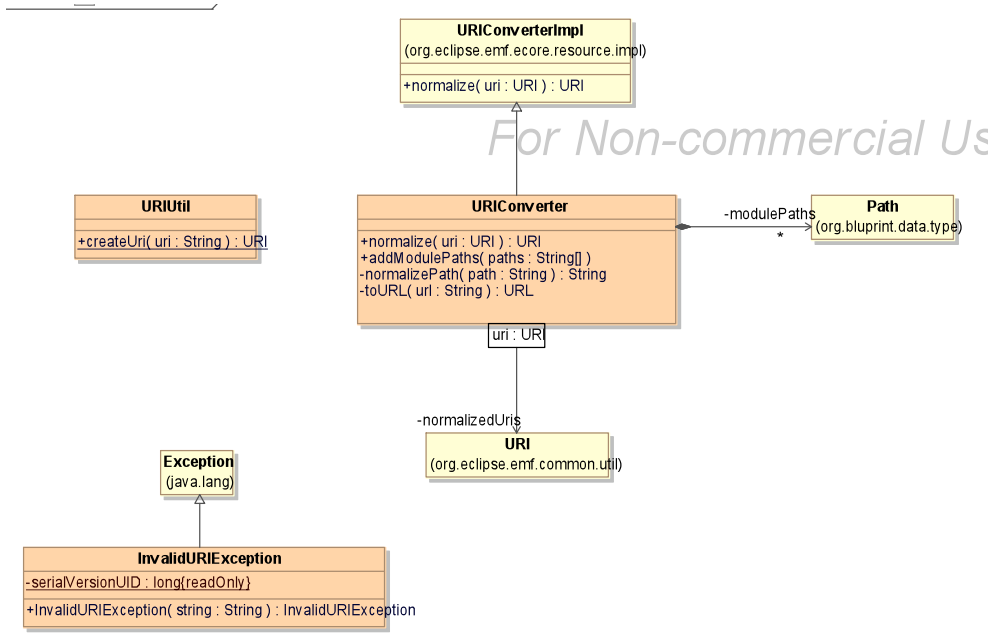


Figure 12 - org.bluprint.app.util

The above are utility classes manipulating URIs.

## org.bluprint.app.util.java


Figure 13 - org.bluprint.app.util.java

The `JavaUtils` class contains a helper function for determining if a string is a Java reserved word.

### org.bluprint.app.util.uml2

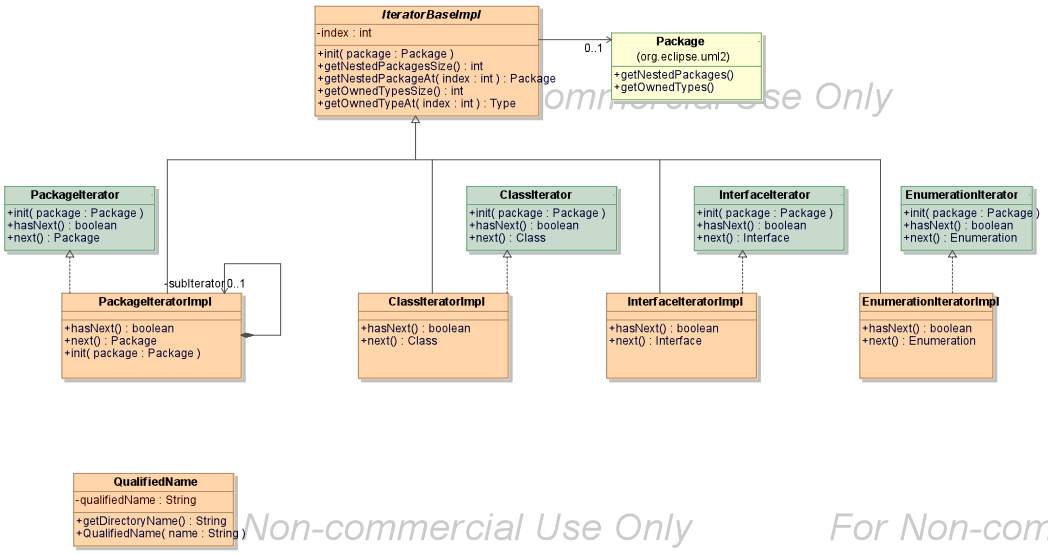




Figure 14 - org.bluprint.app.util.uml2

This package defines a set of interfaces for iterating over elements in an EMF UML2 model together with their implementation.

## Model Layer

### org.bluprint.model.java

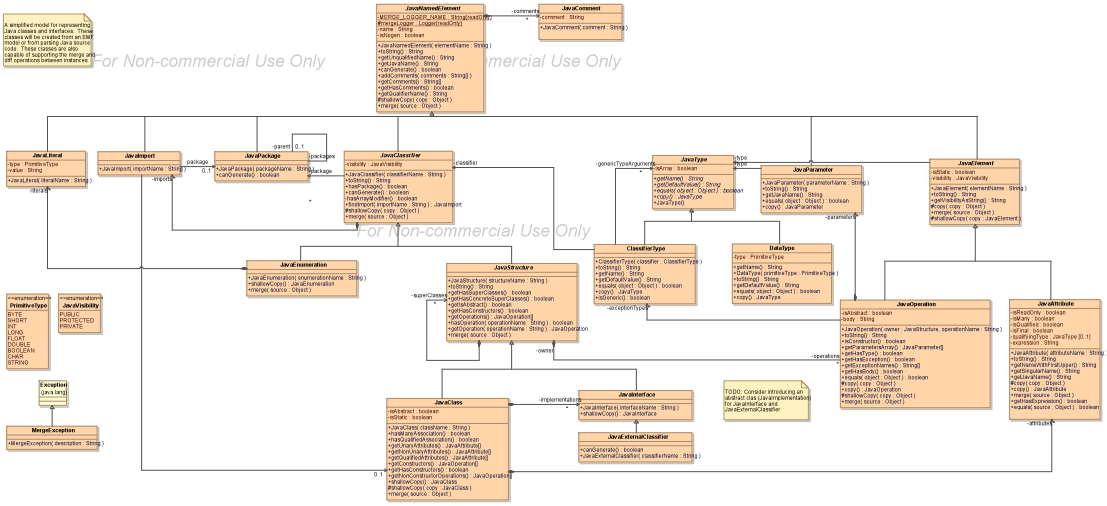



Figure 15 - org.bluprint.model.java

This package contains a simplified model for representing Java classes and interfaces. These classes will be created from an EMF model or from parsing Java source code. These classes are also capable of supporting the merge and diff operations between instances.

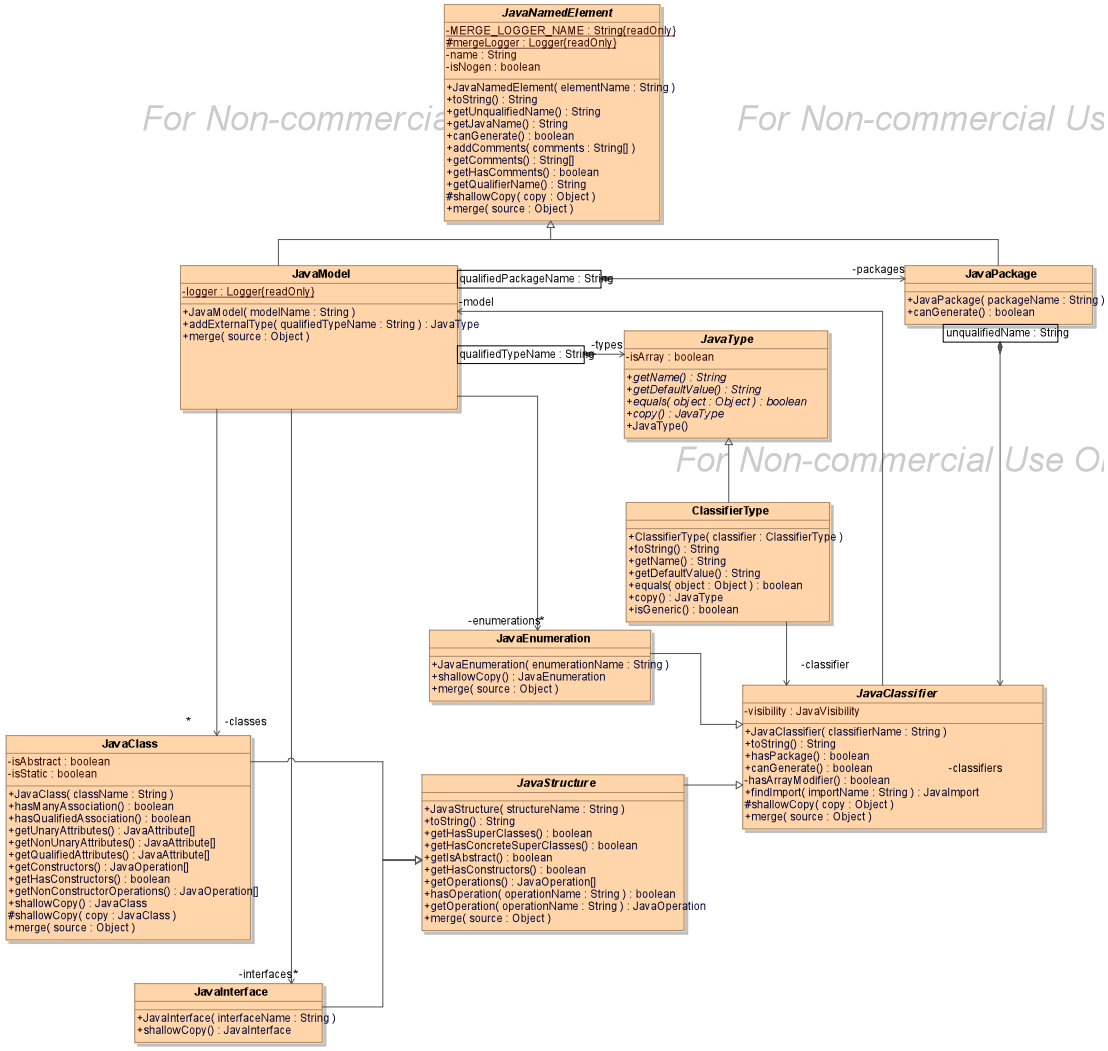


Figure 16 - org.bluprint.model.java – model

The elements of a Java program are held within a `JavaModel`, which contains the root of all classes, interfaces and packages within a Java program.


Figure 17 - org.bluprint.model.java – arrays

As the comment in the model explains, arrays have to be represented as classes ending with the suffix “[]”, so that they can be specified as arguments and the return types for class methods. Bluprint, will not generate code for these classes.