Design Document

# Introduction

Outlined in this document are the considerations and specifications for the end product to be created. We will need to consider the requirements outlined in the requirements analysis and factor these into our designs.

# Environment

We need to consider the environment in which the final product will be designed in and a variety of environmental factors that could affect the development of the project.

## Hardware

The system will need to be able to run on the PCs within the University and will need to run on basic computer hardware with varying specifications.

## Users

The software will need to be fully functional and easy to use, early iterations must focus on usability and accessibility.

## Operating System

For the eclipse plugin to work the machine will need to have eclipse installed which can run on both Microsoft Windows and various Linux distributions. Cross-compatibility will ensure that our plugin can be used by a variety of users, however we need to be cautious as the more platforms are supported the more support we may have to provide.

# Fixed Interfaces

## DBMS

The need for any database management systems have not been identified by the client or within the analysis and design phase. However we could utilise one to track classes and changes for recoverability, this is a consideration that should be made but will most likely not be used in the first full iteration of the plugin.

## Interfaces to other systems

### Eclipse

Our plugin will need to be able to interface with the other components of Eclipse. We will need to be able to gather information on active projects. There will also be a need to connect to the Java Model held by Eclipse on user projects.

### BlueJ

It would be a beneficial, but not essential, function of our plugin to be able to integrate with BlueJ, being able to move projects between BlueJ and Eclipse would make transitioning between the IDEs much simpler for new students.

# Methodology

In this section the chosen methodology for the implementation of the product will be outlined and discussed.

Many methodologies and their spin offs were considered and some of these can be read in the research document.

However the iterative prototype method with elements of scrum was chosen for this group project. This was decided from the risk analysis in the risk document and by the research performed in the research document [Reference to those documents?]

This approach is ideal to this group as the member size is only 2 and therefore adopting a full agile methodology seemed unnecessary as these are tailored for small to medium teams which usually consist of at least 5 to 10 members [Get reference here]. The chosen approach allows for the two members of the group to work on both the implementation and documentation side of the product iteratively, each requirement will be completed and tested before the next requirement is addressed. As there is only 2 people this approach is efficient and easy to maintain [Reference here]. The scrum elements that will be present are a weekly meet up with the client to report on progress and discuss on any topics such as requirements validation.

The nature of the product also requires this methodology, as the knowledge of how to implement a plug in for eclipse is unknown. The iterative prototype approach lets us tackle each requirement separately until it is implemented and tested. As each requirement will involve different functionality that will have to be first researched and learned.

Another reason for this approach is that it allows us to receive feedback during implementation from the client or test sessions held with students. As the product is intended for students it would be wise and useful to use feedback from them and the client after each prototype. With the feedback gained and used after each working prototype it ensures a final product which the client truly wanted.

# System Structure

## Core

The core area of the plugin will comprise of the means to activate the plugin and give access to the required runtimes. The core will handle the connection with the Eclipse PDE and allow for easy integration with the IDE. This area will store details such as user preferences and provide access to the Java Model required for the majority of the system. The core will also provide classes to handle instances and any other representation of user data.

## Perspective

The plugin will need a means to control the perspective to be used in the Eclipse IDE. This part of the system will control the layout of the various views that will be accessible to the user. The perspective will be integrated into the Eclipse IDE and be easily selectable to create an easy to use environment for learning programming and providing an easy means for testing methods with user defined classes.

## Views

The views will make up the main components of the system. Here the main functionality of the plugin will be implemented.

### Class & Package View

The class and package view will provide a means to navigate a user project. In this view the packages and classes will be represented graphically with all of the links between them. This view will also enable users to instantiate an instance of their classes to the object bench. This view will also allow for the creation and deletion of both packages and classes.

### Object Bench

The object bench will allow a user to control instances they have created from the class and package viewer. From the object bench an instance will be made active and methods can be called from a context menu. A selected instance will also be viewable in the inspector.

### Inspector

An inspector view will allow users to view the fields and their values within an instance of their own defined classes. This view should also allow users to edit the values of the respective fields.

## Dialogs

Dialogs will be used to handle the majority of user input. These will facilitate the creation of new classes, new packages and the calling of methods. The dialogs will be very easy to use and will allow users to perform the majority of tasks within the plugin. Dialogs will also be used to inform the user of both information output, errors and return values from the methods called.

## Utilities

The utilities part of the system will contain a set of helper classes with the means to perform common operations within the plugin. This will contain methods to help draw GUI elements for constructors and method calls. Utilities will allow us to gain components to assist in allowing user input and parsing classes created by the users. Within the utilities we will also ensure interaction with the file system of a project to enable manipulation of files within a user created project.

# System Specifications

## Programming Language

The plugin will be created using Java, this is because the primary development platform for the Eclipse IDE is Java.

## Dependencies

## Core Concepts

### Reflection

### SWT

### Java Model (AST)

## Core Functionality

The core contains the main components used to initialise and create an instance of the plugin.

### Activator

The Activator will act in a similar way to a Java class with a main method; it will initialise the plugin and enable the integration with Eclipse.

### Instance

The instance will provide an interface to an instance to be stored on the object bench.

* **Get/Set Fields -** There will be a means to both get the values of fields and set them as required.
* **Get Method -** A means to get a method ready to be called, this will utilise reflection to gain information on parameters and return types.
* **Call Method –** Call a method contained within the instance.
* **Get Field Class –** Returns a representation of a class used by the requested field.
* **Get Class –** The class type of the instance.

### Template

Templates will represent a template class to be placed into a newly created class.

* **Get /Set Name –** The template will have a name reference for quicker loading.
* **Get Template –** Returns the body of the template to be used in creating a new class.

#### Templates

* **Class –** A basic class with example fields, basic constructor and sample method.
* **Abstract Class –** A sample abstract class with easy over-writable methods.
* **Interface –** A sample interface class with a sample method signature.
* **Applet –** An example applet with initialisation and basic GUI setup.
* **Unit Test –** Simple unit test framework.
* **Enumeration –** A basic enum class, sample will show the days of the week.

## Java Model Helper

The Java Model Helper will provide access to the Abstract Syntax Tree (AST) used by Eclipse to represent a user project, along with all of their packages and classes. This unit will be the most used as it will provide all required information on user packages and the classes contained within said classes.

* **Add Class to Loader –** There will need to be a means to utilise a class loader to ensure that the plugin can load and make use of user classes.
* **Get Project –** Obtain the details of a selected project ready for use throughout the plugin. There should also be a means to keep an active project to be able to determine the information to be used in other methods. Through this we should also be able to determine the state of a project (open/closed) and the type of project it is.
* **Get Packages in Project –** Getting the individual packages and their information for use in drawing the packages in the package view and iteration over packages to find certain classes as required.
* **Get Classes in Package –** A means to get a collection of the classes contained within a certain package. This will be useful to obtain the amount of classes in a package, their names, and any other details required from the model.
* **Get Constructors –** Find the constructors of a given class for use in creating instances. The model helper will need to provide the parameters of a constructor, along with the names and types of said parameters**.**
* **Get Underlying Files –** A means to get the underlying files of a package and the classes within, this could be used for a variety of purposes but primarily for adding to the class loader and handling file creation/deletion.
* **New Class/Package** – The facility to create new classes and packages within specified areas of a project.
* **Get Methods of Class –** The Java Model helper will need to be able to iterate over methods within a class and be able to provide details about the method (name, signature), its parameters (both names and types) and its return type.
* **Get a Loaded Class –** To be used with reflection; this functionality will allow an instance to obtain a copy of its parent class to be able to read and write to fields and call methods as required.

## Views

### Class/Package Diagram View

This view provides the user the means to navigate their packages and classes. The majority of actions and functionality from within this section could be implemented through context menus and simple buttons.

* **Add/Remove Package –** It will be critical that a user can both add new packages and remove them from within this view.
* **Add/Remove Class –** It will be useful for the user to be able to add and remove classes from this view.
* **Edit Class –** An action should be present that allows the user to enter an editor by either accessing an option through a context menu or a double click.
* **Instantiate Class –** From this view the user should be able to instantiate a class utilising the variety of available constructors they have devised.
* **Navigation –** The project open should be navigable in this view, double clicks on packages should show the interior classes of the selected package with an option to return to a level above.
* **Draw Classes –** The classes within a package need to be rendered on screen with full inheritance and dependency links in place accurately.

### Object Bench

The object bench controls the instances created within the class diagram view.

* **Remove Instance –** It will be possible to remove instances from the object bench.
* **Selection & Link to Inspector –** When selecting an instance on the object bench the system will make it active within the inspector view. The bench should also highlight a selected instance.
* **Call Methods –** It will be possible to call the methods of an instance from a context menu attached to each instance.
* **Draw Instances -** The instances should be represented graphically in coloured boxes.

### Instance Inspector

The instance inspector shows information on a selected instance.

* **Get/Set Fields –** From the inspector a user will be able to see the fields of an instance and their current value. It should be possible to edit the values of the fields from within the inspector.
* **Array Editor –** The inspector will also be able to edit arrays of primitive types (and maybe more complex types in the future). In editing arrays the user will be able to change the size and update the values in each individual object within.

## Dialogs

The dialogs will be used to gather user input and relay information to the user. The majority of input dialogs should have cancel and confirm buttons to ensure easy navigation.

### Constructor

The constructor dialog is used to instantiate an object based on a constructor passed as an initial variable.

* **Set Parameters –** The dialog will display the required parameters for the specified constructor and these will have appropriate controls that will define the parameters to be passed to the constructor when instantiating the class.

### Method Call

The method call dialog behaves in a highly similar fashion to the constructor dialog only instead of instantiating a class it calls a method within an instance and triggers a return value.

### Method Return

The method return dialog will display a text representation of an object returned by a method called from the object bench.

### New Class

The new class dialog is called when a user requests a new class.

* **Set Class Name –** The dialog will allow a user to define a name for the new class to be created.
* **Select Template –** The template to be used in the class creation is to be set from this dialog.

### New Package

The new package dialog is called when a user requests a new package.

* **Set Package Name –** The name of a new package will be set from within this dialog.

## Utilities

### String Helper

A string helper will be implemented that will assist in handling a variety of strings and inputs as required.

* **Fix Type –** Object types obtained from the AST often don’t match the actual usable name of the class they represent. A method that will fix these text strings into standard class names will be implemented to ensure correct classes are used when required.
* **Strip Extension –** This will be used to strip an extension from a string. This is useful when dealing with class names from the AST, often classes will have the extension attached (.java) which needs to be removed for loading a class.
* **Qualified Name Creation –** The string helper will be able to build a fully qualified name for a class when given the basic name of a class and a full name of the package that contains it. A fully qualified name is required load a class for use by both the plugin and the users own testing.

### Object Utility

The object utility will contain methods to ease repeated actions within the plugin.

* **Linking to Other Components –** The utility will provide the means for views to interact with one another, each view will register with the utility and then the utility will be able to provide required view objects as required.
* **File Handling –** The controls to delete and edit files will be contained within the utility class to reduce the amount of code required to perform these actions at each point they are required.
* **Get Control –** This will take a given type and return a specific control that will accommodate that type. For example if an integer is passed in to the utility it will return a spinner object.
* **Type Handling –** The utility will be able to determine whether a specific type is known and if it is return the correct class associated with the type given.
* **Get/Set Control Values –** Control values can have data set onto them to control the object to be utilised as required. If the control requires a user defined class a combo box will be built based on objects currently on the object bench.
* **Active Instance Control –** The utility will be able to keep track of the currently selected instance and pass it to the inspector if required.

### Template Loader

The template loader is used to load templates from simple text files and convert the text into a Template class for use in creating new classes on user request.

## Noteworthy Algorithms

### Dynamic SWT Controls

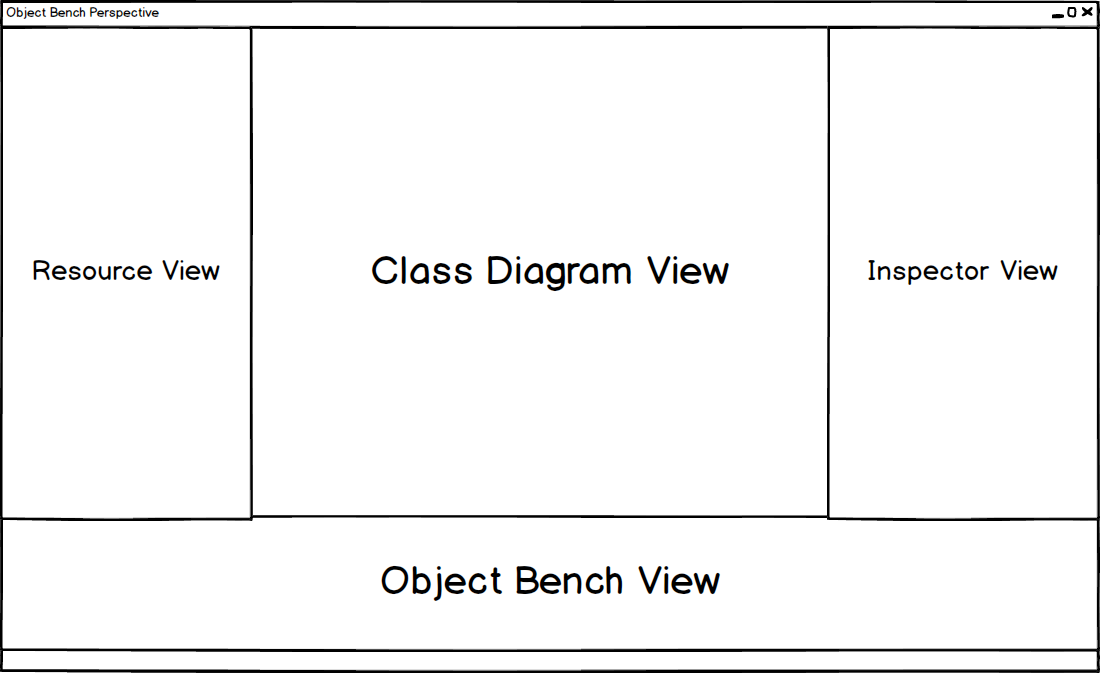
### Sort Classes (Class Diagram)

# GUI Designs

Shown below are the preliminary GUI designs for the plugin.

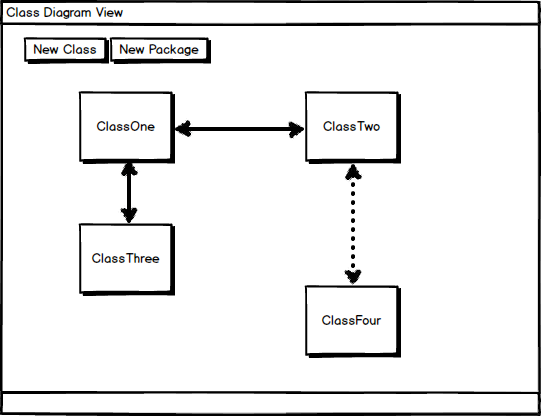
## Perspective

The perspective is the collection of views offered by the plugin. The Resource view is a default Eclipse view that is utilised by the plugin to select the active project.



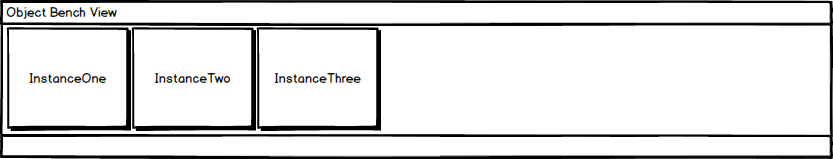
## Class Diagram View

The class diagram view will display the packages and classes within a project. Classes will show inheritance by a solid line and dependency from a staggered line. There will also be buttons for new classes and new packages.



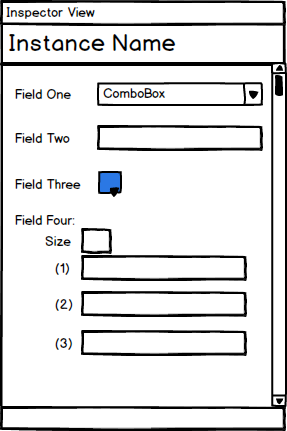
## Object Bench

The object bench will hold instances created from the class diagram view. The instances are represented by coloured buttons. Each button will have a context menu offering the methods from the instance and the means to remove it from the object bench.



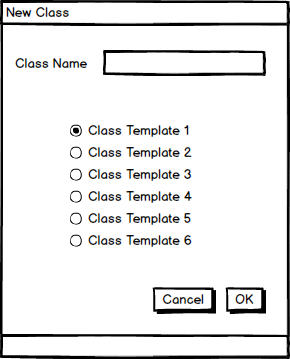
## Inspector View

The inspector view will display a selected instance. The fields will get a required control from the utility class as described above. Arrays will be fully editable with a field offered for size and the fields can be edited as shown by the numbered elements in the design for the inspector.



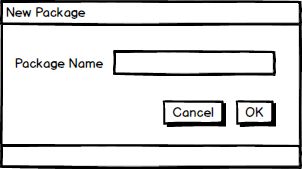
## New Class Dialog

The new class dialog will provide a text box for a class name, radio buttons for the template to be used and buttons for navigation and confirmation.



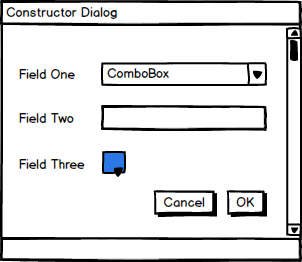
## New Package Dialog

The new package dialog will provide a text box for a package name and buttons for navigation and confirmation.



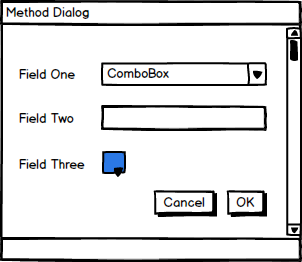
## New Instance Dialog (Constructor)

The new instance dialog will provide controls for the fields required by a constructor and buttons for navigation and confirmation.



## Call Method Dialog

The call method dialog will provide controls for the fields required by a method and the buttons for navigation and confirmation.



# System Models

In this section UML diagrams will be shown in relation to the requirements set out in the requirements analysis document.

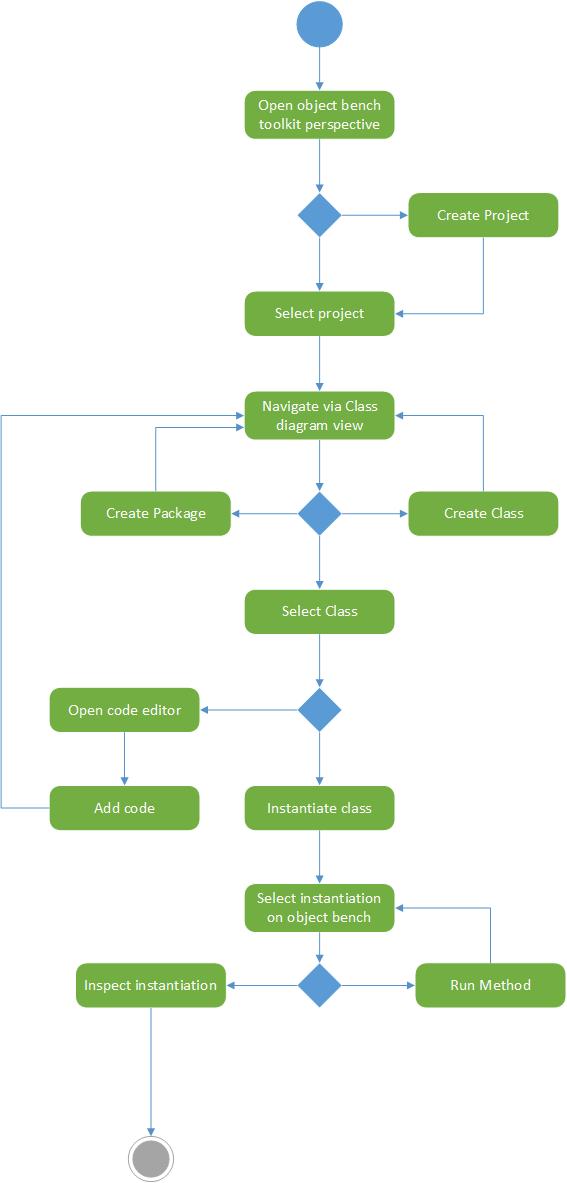
## Activity Diagrams

This diagram is an activity diagram. It shows the process of the user using the system in a step by step fashion; the diagram begins with a start node and ends with an end node. In between these nodes, actions and decisions are outlined.

This version of the activity diagram shows the primary process that the user would take when using the plugin.

After opening the object bench toolkit perspective the user can decide to either create a new java project or select an already existing one. From there the class diagram view is opened, this allows the user to navigate packages and / or classes. The user can either create a package or class or select an already existing class.

From there the user can both open the code editor and add code to that class or instantiate that class. Instantiating the class places the class on the object bench. Selecting this instantiation allows to either inspect it or run a method.

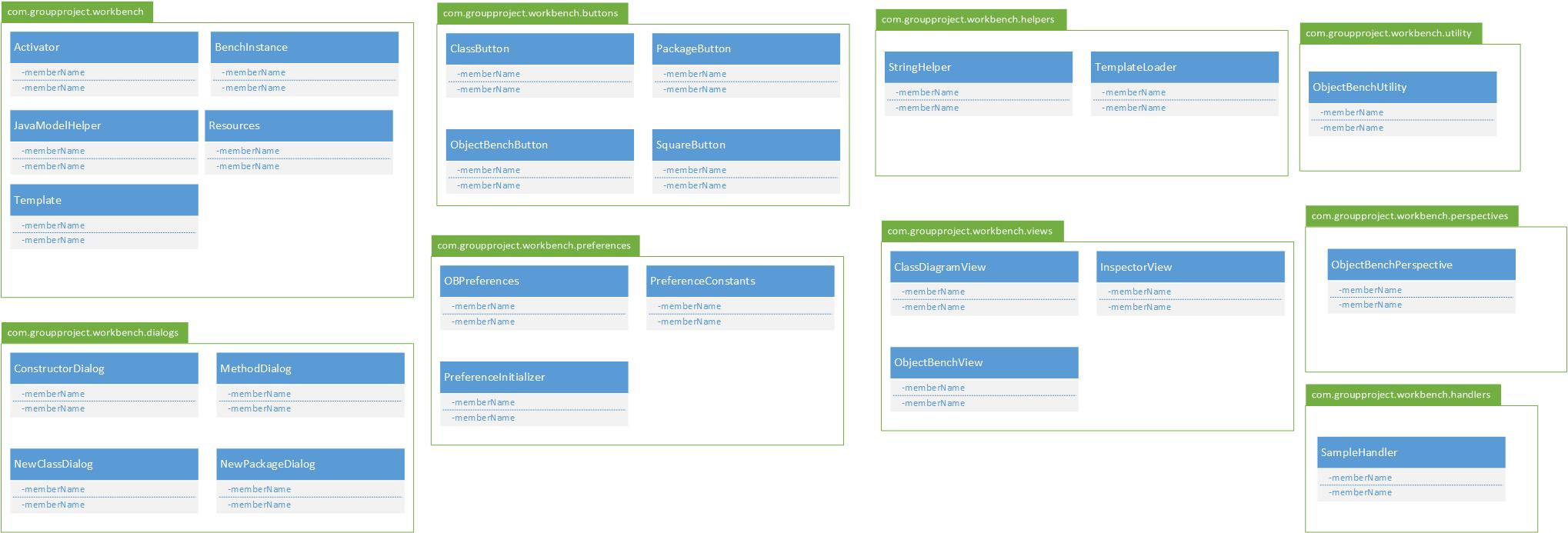


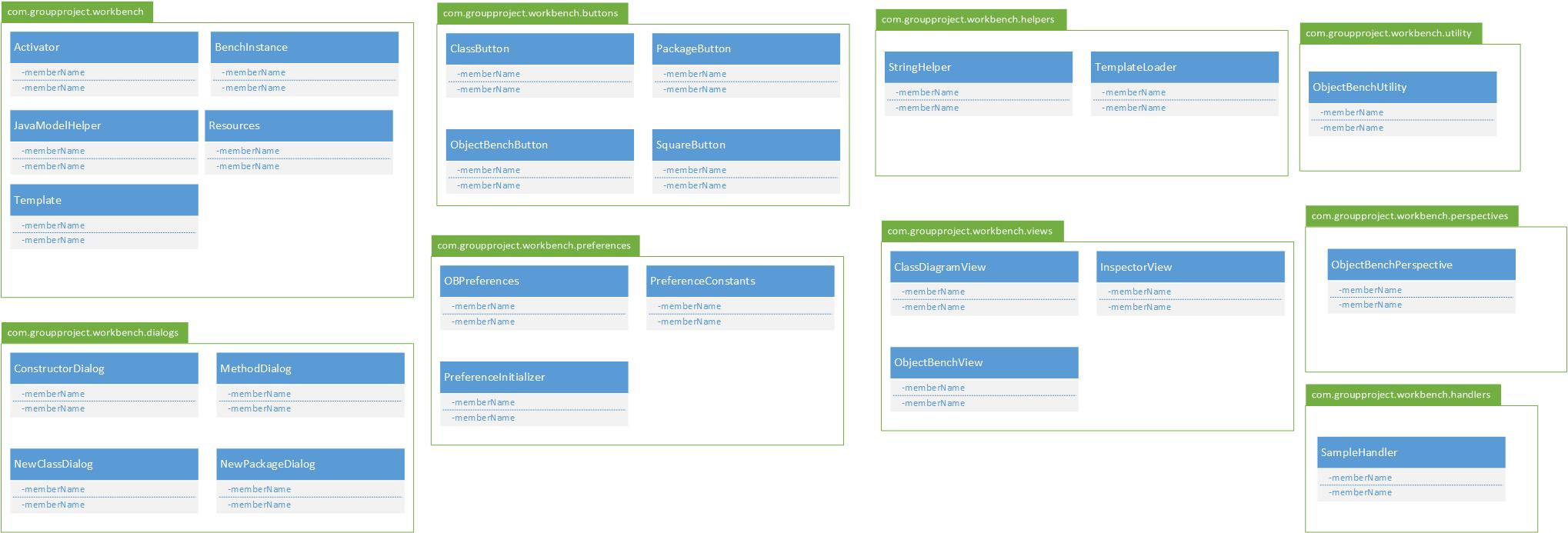
## Use Cases

This diagram is a use case, it shows all the interactions the actor (the user of the plugin in this case) can perform to the system (The Eclipse IDE with the plugin). Here the actor can do the following things:

* Create a new java project
* Select a java project
* Open the perspective of the plugin
* Add a class
* Add a package
* Select a class
* Open the code editor
* Add a method through the code editor
* Instantiate a class
* Selecting the instantiation
* Inspecting the instantiation
* Running a method of the instantiated class
* Running the compiler

## Class Diagram





This diagram shows the structure of the packages and classes of the plugin.

The current version is an early draft as packages and classes may get added and removed.

# Implementation Plan

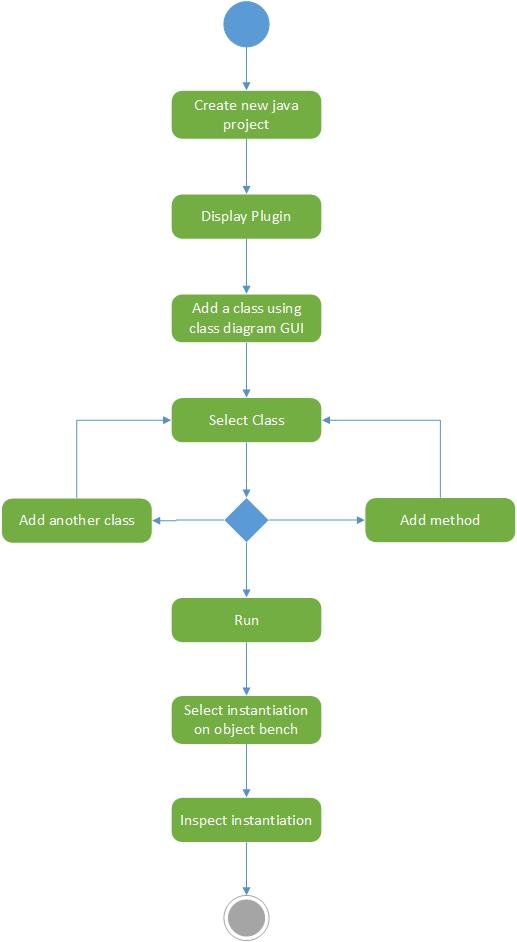
# Evaluation Criteria

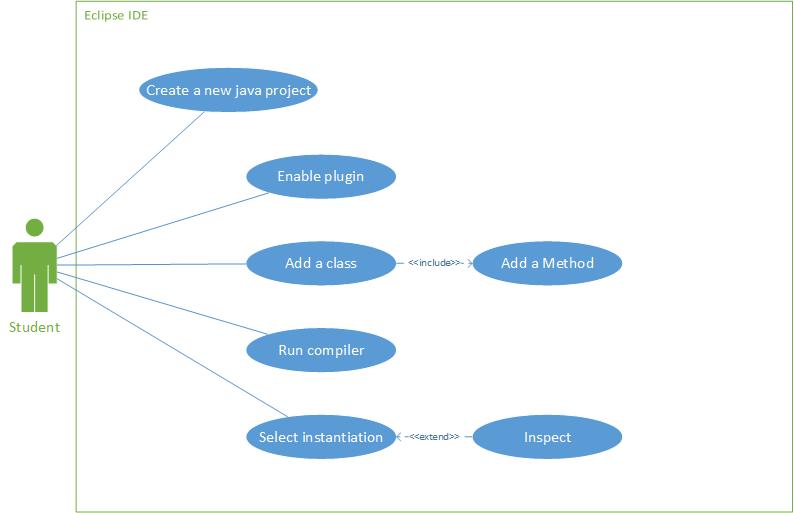
# Conclusion

# Glossary

# References

# Appendix A: Past Diagrams





# Appendix B: Document History

* First created 27/12/2014
* Activity and Use case updated 30/01/2015
* Draft Class Diagram added 7/02/2015