THE JCLASS DESIGNER

Software Requirements Specification

Author: Richard McKenna
Debugging Enterprises™

Based on IEEE Std 830™-1998 (R2009) document format

1 Introduction

The Unified Modeling Language is a standard set of software design diagram formats used for planning in Software Engineering. There are a number of different types of diagrams used but the primary structural design format for Object-Oriented software development is the UML Class Diagram. There are a huge number of modeling tools out there in the market for creating UML Class Diagrams, like VioletUML, but typically, if such a tool can be used to export the diagrams into skeleton code, like .java files, they require purchased licenses.

In this project, the goal is to create a modeling tool specifically to design Java programs using UML Class Diagrams that can then be exported to Java source code. By narrowing the scope of this project to modeling Java programs we can make our tool more useful for Java developers.

1.1 **Purpose**

The purpose of this document is to specify how our ***jClass Modeler*** application should work. This document would then be used as input into this application's design process. The intended audience for this document is all the members of the development team, from the User Interface designers to the Software Engineers and even Salespeople who will ultimately interface with potential customers and advertisers. This document serves as an agreement among all parties and a reference for how the application should ultimately be constructed. Upon completing the reading of this document, one should clearly visualize how it will operate as well as what it ultimately will produce.

1.2 Scope

In this project the goal is to first make an application that Java developers might like to use to design their systems. Therefore, it is important to understand what software designers look for in such projects and provide those features so as to make this application as user friendly as possible. This can only be achieved via an intuitive and carefully organized and attractive user interface. So it is important in this stage that we make sound User Interface design decisions. In addition, it is important that in the next stage i.e. software design, we make sound software design decisions such that our application can be easily extended. The plan is for this application to improve its usability over time for its users. So the plan is to make ***The jClass Designer*** a living piece of software that can easily accommodate change.

1.3 Definitions, acronyms, and abbreviations

Framework – In an object-oriented language, a collection of classes and interfaces that collectively provide a service for building applications or additional frameworks all with a common need.

GUI – Graphical User Interface, visual controls like buttons inside a window in a software application that collectively allow the user to operate the program.

IEEE – Institute of Electrical and Electronics Engineers, the "world's largest professional association for the advancement of technology".

UML – Unified Modeling Language, a standard set of document formats for designing software graphically.

UML Use Case Diagram – A UML document format that specifies how a user will interact with a system. Note that these diagrams do not include technical details. Instead, they are fed as input into the design stage (stage after this one) where the appropriate software designs are constructed based in part on the Use Cases specified in the SRS.

UML Class Diagram – A UML document format that specifies the structural design of an object-oriented software system.

UML Sequence Diagram – A UML document format that specifies the dynamic design of a software system.

1.4 References

IEEE Std 830™-1998 (R2009) – IEEE Recommended Practice for Software Requirements Specification

Wikipedia – Provides adequate definitions for **UML Diagrams**.

1.5 Overview

This SRS will clearly define how ***The jClass Designer*** application should operate. Note that this is not a software design description (SDD), which would deside how to construct the software using UML. This document does not specify how to build the appropriate technologies, it is simply an agreement concerning what to build. Section 2 of this document will provide the context for the project and specify all the conceptual design. Section 3 will present what user interface controls are needed and all program functionality. Section 4 provides a Table of Contents, an Index, and References.

2 Overall description

Each semester in Computer Science classrooms in courses like Computer Science III and Software Engineering, students are required to design software systems before implementing them. This is commonly done for the Java platform and this typically starts with the construction of Class Diagrams. Class diagrams specify all the classes and interfaces to be constructed, what data they'll store, what methods they'll provide, and what the relationships between these things will be. ***The jClass Designer*** is a tool that when completed, will be used for constructing these types of diagrams and then have the ability to export such design to either images or to skeleton source code.

2.1 Product perspective

The UML standard is language independent, meaning it can be used to design Java code, C# code, or whatever object-oriented language one desires. This project will target the Java programming language and by doing so will provide some advantages to the user. Note that most modeling tools can build a variety of UML diagram formats. This tool will only build UML Class Diagrams, which is arguably the most important format, and certainly the most commonly used. As such, this product will need to provide all features available to Java developers in designing code and must do so in user friendly ways. Finally, the application can store its data in all sorts of different ways, but it will be truly useful if it can export useful output like images of the designs and skeleton Java code to help developers start their implementation stages.

2.1.1 System Interfaces

The jClass Designer application will need provide a File Toolbar for managing things like the creation, loading, and saving of designs. The application's workspace will have a Design Component Toolbar on the left that will let the user select different editing modes and structures to add to the diagram. The Design Renderer region will be in the center and is where the class diagram will be rendered and where the user can interact with the various design components via the mouse. When a single class or interface is selected in the existing design, it's editable properties are to be presented for editing in the Component Inspector Toolbar on the right. The UI regions for this application are specified below in **Figure 2.1**. Note that ***The jClass Designer*** will ultimately exist as a standalone Java.

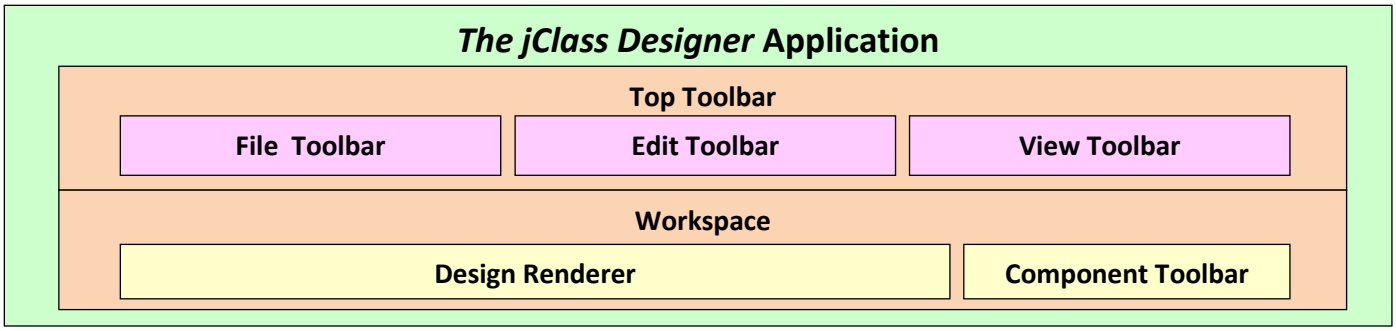


Figure 2.1: The broad screen regions for *The jClass Designer*.

2.1.2 **User Interfaces**

Figure 2.2 below summarizes the ways with which the user will interact with our ***jClass Designer*** application, which will be further detailed using UML Use Case diagrams. These Use-Case diagrams should be fed as input directly into Section 3.1, external interfaces, which is where the design of the user interface is specified. Here is the full list of UML Use-Case Diagrams:

Use Case #	UI Regions	Use Case
2.1	File Toolbar	New jClass Design
2.2	File Toolbar	Load jClass Design
2.3	File Toolbar	Save jClass Design
2.4	File Toolbar	Save As jClass Design
2.5	File Toolbar	Export jClass Design to Photo
2.6	File Toolbar	Export jClass Design to Code
2.7	File Toolbar	Exit
2.8	Edit Toolbar & Design Renderer	Select and Move Class/Interface
2.9	Edit Toolbar & Design Renderer	Resize Class/Interface
2.10	Edit Toolbar & Design Renderer	Add Class
2.11	Edit Toolbar & Design Renderer	Add Interface
2.12	Edit Toolbar & Design Renderer	Remove Class/Interface
2.13	Edit Toolbar	Undo
2.14	Edit Toolbar	Redo
2.15	View Toolbar	Zoom In
2.16	View Toolbar	Zoom Out
2.17	View Toolbar	Enable/Disable Grid Rendering
2.18	View Toolbar	Enable/Disable Grid Snapping
2.19	Design Renderer	Split Line Segment
2.20	Design Renderer	Remove Line Segment Point
2.21	Design Renderer	Move Line Segment Point
2.22	Component Toolbar	Edit Class/Interface Name
2.23	Component Toolbar	Edit Package Name
2.24	Component Toolbar	Edit Parent Class/Interface
2.25	Component Toolbar	Select Variable
2.26	Component Toolbar	Add Variable
2.27	Component Toolbar	Edit Variable
2.28	Component Toolbar	Remove Variable
2.29	Component Toolbar	Select Method
2.30	Component Toolbar	Add Method
2.31	Component Toolbar	Edit Method
2.32	Component Toolbar	Remove Method

Figure 2.2: Overview of Use-Case Diagrams

Use Case 2.1: New jClass Design

Use-Case:	New jClass Design
Primary Actor:	Software Designer
Goal in Context:	The user may at any point create a new design to work on. Note that the application can only edit one design at a time.
Preconditions:	The application has been started and initialized
Scenario:	**1.** User starts the application, which loads app settings **2.** User clicks on the "New jClass Design" button
Exceptions:	This button should always be enabled in the file toolbar. Note that should an design already be loaded and unsaved, the application should prompt the user to save first.
Priority:	Essential, must be implemented
When available:	First Benchmark
Frequency of use:	Used very often, at least once for each design
Open Issues:	Style of button should be finalized by UI designer. Should use an elegant, descriptive icon that suits application theme.

Use Case 2.2: Load jClass Design

Use-Case:	Load jClass Design
Primary Actor:	Software Designer
Goal in Context:	The user may at any point load an existing design to work on. Note that the application can only edit one design at a time.
Preconditions:	A design exists as a file that can be loaded.
Scenario:	1. User starts the application, which loads app settings 2. User clicks on the "Load jClass Design" button 3. User then selects the saved design to load 4. jClass then loaded for editing
Exceptions:	This button should always be enabled in the file toolbar. Note that should a design already be loaded and unsaved, the application should prompt the user to save first.
Priority:	Essential, must be implemented
When available:	Second Benchmark
Frequency of use:	Used very often, many times per design.
Open Issues:	Style of button should be finalized by UI designer. Should use an elegant, descriptive icon that suits application theme.

Use Case 2.3: Save jClass Design

Use-Case:	Save jClass Design
Primary Actor:	Software Designer
Goal in Context:	The user may at any point save the current design being worked on, as long as one is in progress. Note that the application can only edit one design at a time.
Preconditions:	The application has been started and initialized and either a new design has been started or an existing one has been loaded
Scenario:	1. User starts the application, which loads app settings 2. User either creates a new design or loads an existing one 3. User edits various design details 4. User clicks on "Save jClass Design" button 5. If first time saving, user is prompted for jClass Design file name 6. Design is saved to file using proper file name
Exceptions:	This button should always be provided in the file toolbar, but should only be enabled if the design has been edited since the last save.
Priority:	Essential, must be implemented
When available:	Second Benchmark
Frequency of use:	Used very often, many times per jClass Design.
Open Issues:	Style of button should be finalized by UI designer. Should use an elegant, descriptive icon that suits application theme.

Use Case 2.4: Save As jClass Design

Use-Case:	Save As jClass Design
Primary Actor:	Software Designer
Goal in Context:	The user may at any point save the current jClass Design being worked under a different file name, as long as one is in progress. Note that the application can only edit one design at a time.
Preconditions:	The application has been started and initialized and an existing jClass Design has been loaded
Scenario:	1. User starts the application, which loads app settings 2. User loads an existing jClass Design 3. User edits various design details 4. User clicks on "Save As jClass Design" button 5. User is prompted for design file name 6. Design is saved to file using proper file name
Exceptions:	This button should always be provided in the file toolbar, but should only be enabled if a design is currently being edited.
Priority:	Essential, must be implemented
When available:	Second Benchmark
Frequency of use:	Used at most once per design
Open Issues:	Style of button should be finalized by UI designer. Should use an elegant, descriptive icon that suits application theme.

Use Case 2.5: Export jClass Design to Photo

Use-Case:	Export jClass Design to Photo
Primary Actor:	Software Designer
Goal in Context:	Export current design to a custom named image file.
Preconditions:	The application has started and the user edited a jClass Design. This would typically be done when a design is completed, but could be done prior as well
Scenario:	1. User is editing a jClass Design, entering content, etc. 2. User clicks on "Export jClass Design to Photo" button 3. Program prompts user for image name and location to save file. 4. Program generates image of full design and exports to image file.
Exceptions:	This button should always be provided in the file toolbar and should only be enabled if a design is currently being edited.
Priority:	Essential, must be implemented
When available:	Second Benchmark
Frequency of use:	Used at least once per design.
Open Issues:	Style of button should be finalized by UI designer. Should use an elegant, descriptive icon that suits application theme.

Use Case 2.6: Export jClass Design to Code

Use-Case:	Export jClass Design to Code
Primary Actor:	Software Designer
Goal in Context:	Export current design to skeleton .java code
Preconditions:	The application has started and the user edited a jClass Design. This would typically be done when a design is completed, but could be done prior as well
Scenario:	1. User is editing a jClass Design, entering content, etc. 2. User clicks on "Export jClass Design to Code" button 3. Program prompts user for directory location to put generated code 4. Program generates directories for all packages using proper directory structure (including subpackages) and .java files for all designed classes Note that the code that is generated will compile as all methods will return dummy values to satisfy the compiler.
Exceptions:	This button should always be provided in the file toolbar and should only be enabled if a design is currently being edited.
Priority:	Essential, must be implemented
When available:	Second Benchmark
Frequency of use:	Used at least once per design.
Open Issues:	Style of button should be finalized by UI designer. Should use an elegant, descriptive icon that suits application theme.

Use Case 2.7: Exit

Use-Case:	Exit
Primary Actor:	Software Designer
Goal in Context:	The user wishes to close the application
Preconditions:	The user may wish to do so at any point, so the program must be prepared to handle the exit function at any time.
Scenario:	1. User is using application in any possible way 2. User clicks on "Exit" button 3. If a design is unsaved, prompt user to save 4. Exit application
Exceptions:	N/A
Priority:	Essential, must be implemented
When available:	Second Benchmark
Frequency of use:	More than once per jClass Design.
Open Issues:	Style of button should be finalized by UI designer. Should use an elegant, descriptive icon that suits application theme.

Use Case 2.8: Select and Move Class/Interface

Use-Case:	Select and Move Class/Interface
Primary Actor:	Software Designer
Goal in Context:	While editing a design, the user may need to select an element within the design for further editing or for changing how it appears.
Preconditions:	The app has started and the user is editing a design.
Scenario:	1. User is editing a design 2. User clicks on the "Selection Tool" button, which should change the cursor to an arrow selector 3. The user then clicks on a Class or Interface box in the Design Renderer 4. The selected Class or Interface box should then be highlighted to denote its selection 5. Once selected the class/interface can be dragged to move it to a desired location
Exceptions:	These Selection Tool button should only be enabled when the selection tool is not currently in use.
Priority:	Essential, must be implemented
When available:	First Benchmark
Frequency of use:	Many times per design. Needed in order to select a class or interface for editing, moving, or removing.
Open Issues:	Style of button and box highlighting effect should be finalized by UI designer. Should use an elegant, descriptive icon that suits application theme.

Use Case 2.9: Resize Class/Interface

Use-Case:	Resize Class/Interface
Primary Actor:	Software Designer
Goal in Context:	During editing, a user may wish to change the size of a class or interface box.
Preconditions:	The application has been started and a design is being edited with at least one class or interface.
Scenario:	1. User is editing a design that has at least one class or interface 2. User selects a Class or Interface box to resize. 3. User mouses over edge of Class or Interface box which changes the cursor to a resize cursor 4. User presses mouse button on edge of Class or Interface box and drags edge to resize 5. User releases mouse button to leave Class or Interface box at size
Exceptions:	If there are no classes or interfaces in the design there is nothing to resize
Priority:	Essential, must be implemented
When available:	Third Benchmark
Frequency of use:	Many times per design
Open Issues:	N/A

Use Case 2.10: Add Class

Use-Case:	Add Class
Primary Actor:	Software Designer
Goal in Context:	While editing a design this Use Case is for when the user wishes to go to add a Class to the design
Preconditions:	Design editing is underway
Scenario:	**1.** User is editing a jClass Design **2.** User clicks on "Add Class" button **3.** A new box appears in the design with a default class name and with it selected by default and thus its properties loaded into the component toolbar controls, with no variables and no methods
Exceptions:	N/A
Priority:	Essential, must be implemented
When available:	First Benchmark
Frequency of use:	Many times per session for every session
Open Issues:	Style of button should be finalized by UI designer. Should use an elegant, descriptive icon that suits application theme.

Use Case 2.11: Add Interface

Use-Case:	Add Interface
Primary Actor:	Software Designer
Goal in Context:	While editing a design this Use Case is for when the user wishes to go to add an Interface to the design
Preconditions:	Design editing is underway
Scenario:	1. User is editing a jClass Design 2. User clicks on "Add Interface" button 3. A new box appears in the design with a default interface name and with it selected by default and thus its properties loaded into the component toolbar controls, with no methods
Exceptions:	N/A
Priority:	Essential, must be implemented
When available:	Third Benchmark
Frequency of use:	Many times per session for every session
Open Issues:	Style of button should be finalized by UI designer. Should use an elegant, descriptive icon that suits application theme.

Use Case 2.12: Remove Class/Interface

Use-Case:	Remove Class/Interface
Primary Actor:	Software Designer
Goal in Context:	The user will need a way to remove existing classes and interfaces from a design
Preconditions:	A design is being edited and it has at least one class or interface box.
Scenario:	1. User is editing a jClass Design 2. User clicks on "Add Interface" button 3. A new box appears in the design with a default interface name and with it selected by default and thus its properties loaded into the component toolbar controls, with no methods
Exceptions:	N/A
Priority:	Essential, must be implemented
When available:	Third Benchmark
Frequency of use:	Many times per design
Open Issues:	Style of button should be finalized by UI designer. Should use an elegant, descriptive icon that suits application theme.

Use Case 2.13: Undo

Use-Case:	Undo
Primary Actor:	Software Designer
Goal in Context:	The user wishes to undo their most recent action
Preconditions:	The user is editing a design and some action has been taken to change the design
Scenario:	**1.** User is editing a design **2.** User has edited the design in some way **3.** User clicks on "Undo" button and the most recent edit is undone
Exceptions:	All actions must have the ability to be undone.
Priority:	Essential, must be implemented
When available:	Third Benchmark
Frequency of use:	Many times per design
Open Issues:	Style of button should be finalized by UI designer. Should use an elegant, descriptive icon that suits application theme.

Use Case 2.14: Redo

Use-Case:	Redo
Primary Actor:	Software Designer
Goal in Context:	The user wishes to redo their most recently undone action
Preconditions:	The user is editing a design and has undone some action or series of actions and now wishes to redo one or many of those undone actions
Scenario:	1. User is editing a design 2. User undoes one or more recent actions that were done in order 3. User clicks on "Redo" button one or more times and the most recent undo or series of undo actions are redone
Exceptions:	All actions must have the ability to be undone and then also redone.
Priority:	Essential, must be implemented
When available:	Third Benchmark
Frequency of use:	Many times per design
Open Issues:	Style of button should be finalized by UI designer. Should use an elegant, descriptive icon that suits application theme.

Use Case 2.15: Zoom In

Use-Case:	Zoom In
Primary Actor:	Software Designer
Goal in Context:	The user wishes to see a closer view of the current design
Preconditions:	User is editing a design that has at least one class or interface
Scenario:	1. User is editing a design 2. User clicks on the "Zoom In" button, which zooms in the view of the design on the Design Renderer x 2
Exceptions:	Zooming in should only be done to some reasonable limit
Priority:	Essential, must be implemented
When available:	Third Benchmark
Frequency of use:	Many times per design
Open Issues:	Style of button should be finalized by UI designer. Should use an elegant, descriptive icon that suits application theme.

Use Case 2.16: Zoom Out

Use-Case:	Zoom Out
Primary Actor:	Software Designer
Goal in Context:	The user wishes to see a broader view of the current design
Preconditions:	User is editing a design that has at least one class or interface
Scenario:	1. User is editing a design 2. User clicks on the "Zoom Out" button, which zooms in the view of the design on the Design Renderer x ½
Exceptions:	Zooming out should only be done to some reasonable limit
Priority:	Essential, must be implemented
When available:	Third Benchmark
Frequency of use:	Many times per design
Open Issues:	Style of button should be finalized by UI designer. Should use an elegant, descriptive icon that suits application theme.

Use Case 2.17: Enable/Disable Grid Rendering

Use-Case:	Enable/Disable Grid Rendering
Primary Actor:	Software Designer
Goal in Context:	Grid lines can help the designer view boxes and properly line them up to make their design neater. This will allow the user to enable/disable the background rendering of grid lines in the Design Renderer.
Preconditions:	The user is editing a design.
Scenario:	1. User is editing a design and so viewing a design in the Design Renderer 2. User clicks the Grid checkbox, which either enables or disables the rendering of gridlines in the Design Renderer
Exceptions:	Note that grid lines should still be rendered properly when the view is at a different zoom level meaning the grid lines should zoom in and out as well.
Priority:	Essential, must be implemented
When available:	Third Benchmark
Frequency of use:	More than once per design.
Open Issues:	Style of checkbox and text should be finalized by UI designer. Note that the grid lines should not overwhelm the design.

Use Case 2.18: Enable/Disable Grid Snapping

Use-Case:	Enable/Disable Grid Snapping
Primary Actor:	Software Designer
Goal in Context:	Grid lines can help the designer line up boxes and by having snap enabled all class and interface boxes will be aligned along their left edge with grid lines.
Preconditions:	The user is editing a design.
Scenario:	1. User is editing a design and so viewing a design in the Design Renderer 2. User clicks the Snap checkbox, which either enables or disables the snapping of class and interface box left edges to grid lines in the Design Renderer
Exceptions:	Note that once enabled, all subsequent sizing and moving and adding of boxes should snap, but snapping should not work retroactively.
Priority:	Essential, must be implemented
When available:	Third Benchmark
Frequency of use:	More than once per design.
Open Issues:	Style of checkbox and text should be finalized by UI designer.

Use Case 2.19: Split Line Segment

Use-Case:	Split Line Segment
Primary Actor:	Software Designer
Goal in Context:	Line segments will represent relationships between classes and interfaces, the user will want to further segment these lines in order to customize the drawing
Preconditions:	The design must have at least two boxes with at least one relationship connector
Scenario:	1. User is editing a design with at least two boxes with a relationship 2. User selects a line segment by clicking on it 3. User presses 'S' key to split the line segment, which adds a point to its center. Note that points in all line segments are selectable when the line segment is selected
Exceptions:	All line segments should be able to be split
Priority:	Essential, must be implemented
When available:	Third Benchmark
Frequency of use:	Many times per design.
Open Issues:	Style of line segment is up to UI designer, but should be viewable in diagrams.

Use Case 2.20: Remove Line Segment Point

Use-Case:	Remove Line Segment Point
Primary Actor:	Software Designer
Goal in Context:	Line segments will represent relationships between classes and interfaces, the user will want to further segment these lines in order to customize the drawing and so will need a means to merge line segments as well.
Preconditions:	The design must have at least two boxes with at least one relationship connector that has is actually multiple line segments
Scenario:	1. User is editing a design with at least two boxes with a relationship that has two line segments 2. User selects a point connecting two line segments by clicking on it 3. User presses 'M' key to merge the line segments, meaning it deletes the selected point, which should connect the other two points making two line segments one. Note that points in all line segments are selectable when the line segment is selected
Exceptions:	This can only be done if a connector has at least two line segments
Priority:	Essential, must be implemented
When available:	Third Benchmark
Frequency of use:	Many times per design
Open Issues:	Style of line segment is up to UI designer, but should be viewable in diagrams.

Use Case 2.21 Move Line Segment Point

Use-Case:	Move Line Segment Point
Primary Actor:	Software Designer
Goal in Context:	Line segments will represent relationships between classes and interfaces, the user will want to edit these to improve the look of their design, this includes moving line segments about.
Preconditions:	The design must have at least two boxes with at least one relationship connector.
Scenario:	1. User is editing a design with at least two boxes with a relationship. 2. User selects a point on a line segment. 3. User drags the point to where they like. Note that lines should not be dragged off of one of the class/interface boxes if they are at such an end.
Exceptions:	Points can be free moved only if they are not attached to a class/interface box. For those points, they can be moved, but just around such a box.
Priority:	Essential, must be implemented
When available:	Third Benchmark
Frequency of use:	Many times per design
Open Issues:	Style of points is up to UI designer, but should be viewable in diagrams.

Use Case 2.22: Edit Class/Interface Name

Use-Case:	Edit Class/Interface Name
Primary Actor:	Software Designer
Goal in Context:	The user has selected a Class/Interface and would like to change its name
Preconditions:	The user is editing a design with at least one Class/Interface
Scenario:	1. User is editing a design with at least one class/interface 2. User selects a Class/Interface box, which loads that element's info into the Component Toolbar controls 3. User updates name in text field to change class/interface name
Exceptions:	Should not allow entry of name of class that already exists in current design with same package.
Priority:	Essential, must be implemented
When available:	First Benchmark
Frequency of use:	Many times per design
Open Issues:	Style of label and text field should be finalized by UI designer.

Use Case 2.23: Edit Package Name

Use-Case:	Edit Package Name
Primary Actor:	Software Designer
Goal in Context:	The user has selected a Class/Interface and would like to change its package
Preconditions:	The user is editing a design with at least one Class/Interface
Scenario:	1. User is editing a design with at least one class/interface 2. User selects a Class/Interface box, which loads that element's info into the Component Toolbar controls 3. User updates text in text field to change class/interface package name
Exceptions:	Should not allow entry of package name if its package/class combo already exists in design
Priority:	Essential, must be implemented
When available:	First Benchmark
Frequency of use:	Many times per design
Open Issues:	Style of label and text field should be finalized by UI designer.

Use Case 2.24: Edit Parent Class/Interface

Use-Case:	Edit Parent Class/Interface
Primary Actor:	Software Designer
Goal in Context:	The user has selected a Class/Interface and would like to change its parent class/interface
Preconditions:	The user is editing a design with at least one Class/Interface
Scenario:	1. User is editing a design with at least one class/interface 2. User selects a Class/Interface box, which loads that element's info into the Component Toolbar controls 3. User selects a Parent Class/Interface from a drop down box containing classes in the current design that also allows the user to type in new classes/interfaces 4. Upon selection of a Parent other that Object, additional Parents should be selectable to account for multiple Interfaces.
Exceptions:	We will not enforce whether a parent is a class or interface, so the user will have to police that, but our application must enforce the fact that it can accommodate multiple parents where the assumption is that only one is a class.
Priority:	Essential, must be implemented
When available:	Third Benchmark
Frequency of use:	Many times per design
Open Issues:	Style of combo box is left to UI designer.

Use Case 2.25: Select Variable

Use-Case:	Select Variable
Primary Actor:	Software Designer
Goal in Context:	The user has selected a Class/Interface and would like to select one of its variables so that it can be removed from the design
Preconditions:	The user is editing a design with at least one Class/Interface that has a variable
Scenario:	1. User is editing a design with at least one class/interface 2. User selects a Class/Interface box, which loads that element's info into the Component Toolbar controls, including its variables 3. User clicks on one of its variables, which should highlight its row
Exceptions:	Note that some classes may not have variables
Priority:	Essential, must be implemented
When available:	Third Benchmark
Frequency of use:	Many times per design
Open Issues:	Style of variable controls in row should be finalized by UI designer.

Use Case 2.26: Add Variable

Use-Case:	Add Variable
Primary Actor:	Software Designer
Goal in Context:	The user has selected a Class/Interface and would like to add a variable
Preconditions:	The user is editing a design with at least one Class/Interface
Scenario:	1. User is editing a design with at least one class/interface 2. User selects a Class/Interface box, which loads that element's info into the Component Toolbar controls 3. User clicks on the Add Variable button which should add a row with default values
Exceptions:	Default name for new variable should not match name of any existing variable in class/interface
Priority:	Essential, must be implemented
When available:	Third Benchmark
Frequency of use:	Many times per design
Open Issues:	Style of variable controls in row should be finalized by UI designer.

Use Case 2.27: Edit Variable

Use-Case:	Edit Variable
Primary Actor:	Software Designer
Goal in Context:	The user has selected a Class/Interface and would like to edit one of its variables
Preconditions:	The user is editing a design with at least one Class/Interface that has a variable
Scenario:	1. User is editing a design with at least one class/interface with a variable 2. User selects a Class/Interface box, which loads that element's info into the Component Toolbar controls, including its variables 3. User edits the variable including its name and type as well as whether its static or not and its accessibility modifier.
Exceptions:	The application should make sure types are legal types
Priority:	Essential, must be implemented
When available:	Third Benchmark
Frequency of use:	Many times per design
Open Issues:	Style of variable controls in row should be finalized by UI designer.

Use Case 2.28: Remove Variable

Use-Case:	Remove Variable
Primary Actor:	Software Designer
Goal in Context:	The user has selected a Class/Interface and would like to remove one of its variables
Preconditions:	The user is editing a design with at least one Class/Interface that has a variable
Scenario:	1. User is editing a design with at least one class/interface 2. User selects a Class/Interface box, which loads that element's info into the Component Toolbar controls, including its variables 3. User selects a variable 4. User clicks on "Remove Variable" button which should remove it from the class
Exceptions:	Note that some classes/interfaces may not have variables
Priority:	Essential, must be implemented
When available:	Third Benchmark
Frequency of use:	Many times per design
Open Issues:	Style of variable controls in row should be finalized by UI designer.

Use Case 2.29: Select Method

Use-Case:	Select Method
Primary Actor:	Software Designer
Goal in Context:	The user has selected a Class/Interface and would like to select one of its methods so that it can be removed from the design
Preconditions:	The user is editing a design with at least one Class/Interface that has a method
Scenario:	1. User is editing a design with at least one class/interface 2. User selects a Class/Interface box, which loads that element's info into the Component Toolbar controls, including its methods 3. User clicks on one of its methods, which should highlight its row
Exceptions:	Note that some classes may not have methods
Priority:	Essential, must be implemented
When available:	Third Benchmark
Frequency of use:	Many times per design
Open Issues:	Style of method controls in row should be finalized by UI designer.

Use Case 2.30: Add Method

Use-Case:	Add Method
Primary Actor:	Software Designer
Goal in Context:	The user has selected a Class/Interface and would like to add a method
Preconditions:	The user is editing a design with at least one Class/Interface
Scenario:	1. User is editing a design with at least one class/interface 2. User selects a Class/Interface box, which loads that element's info into the Component Toolbar controls 3. User clicks on the Add Method button which should add a row with default values
Exceptions:	Default signature for new method should not match signature of any existing method in class/interface
Priority:	Essential, must be implemented
When available:	Third Benchmark
Frequency of use:	Many times per design
Open Issues:	Style of method controls in row should be finalized by UI designer.

Use Case 2.31: Edit Method

Use-Case:	Edit Method
Primary Actor:	Software Designer
Goal in Context:	The user has selected a Class/Interface and would like to edit one of its methods
Preconditions:	The user is editing a design with at least one Class/Interface that has a method
Scenario:	1. User is editing a design with at least one class/interface with a method 2. User selects a Class/Interface box, which loads that element's info into the Component Toolbar controls, including its methods 3. User edits the method including its name return type and method arguments as well as whether its static or not and its accessibility modifier.
Exceptions:	The application should make sure types are legal types
Priority:	Essential, must be implemented
When available:	Third Benchmark
Frequency of use:	Many times per design
Open Issues:	Style of method controls in row should be finalized by UI designer.

Use Case 2.32: Remove Method

Use-Case:	Remove Method
Primary Actor:	Software Designer
Goal in Context:	The user has selected a Class/Interface and would like to remove one of its methods
Preconditions:	The user is editing a design with at least one Class/Interface that has a method
Scenario:	1. User is editing a design with at least one class/interface 2. User selects a Class/Interface box, which loads that element's info into the Component Toolbar controls, including its methods 3. User selects a method 4. User clicks on "Remove Method" button which should remove it from the class
Exceptions:	Note that some classes/interfaces may not have methods
Priority:	Essential, must be implemented
When available:	Third Benchmark
Frequency of use:	Many times per design
Open Issues:	Style of method controls in row should be finalized by UI designer.

2.1.3 **Hardware Interfaces**

The target platform is the PC. This application requires a lot of textual input, which means we should target devices with keyboards like computers.

2.1.4 **Software Interfaces**

The jClass Designer will be developed using the Java language. Note that since this may be our version of many, we should be careful in designing our classes to make sure they can accommodate future revisions. As part of this, it is important to manage the program data independent of its presentation. The application should use Java's JavaFX framework for building user interfaces as it is best suited to accommodating rendered content.

2.1.5 **Communications Interfaces**

Note that in this product version the application will generate Java source code that should compile. It should also organize classes into their proper package directories. Note that the generated source code should contain methods that looks good, meaning it should be properly structured with newlines and indentation and should also have dummy return values to satisfy the compiler.

2.1.6 **Memory Constraints**

Since this application can be used for making any number of designs but such diagrams are not likely to require large amounts of memory.

2.1.7 **Operations**

It is important to note that making a jClass design is a fluid process by which the user may continually change their mind regarding what to include. Therefore any action for adding and editing a design must be undoable.

2.1.8 **Site Adaptation Requirements**

N/A

2.2 **Product functions**

It is the goal of this application to be intuitive and user friendly, so actions must work in user friendly ways that the user expects.

2.3 **User characteristics**

This application works on the assumption that the user knows a bit about software design. It will provide some efficiencies in terms of making it easier for the user to do certain things, but ultimately the user still has much control and can make both good and bad design decisions.

2.4 **Constraints**

It is important that this application prevent the user from making bad decisions when possible. This is not always possible, but should enforce rules when appropriate, like in the selection of variable, method argument, and method return types. Limiting choice in these controls through combo boxes will help employ foolproof design principles.

2.5 **Assumptions and dependencies**

It is assumed that the user is only interested in Java programs. No other languages will be considered. Note that this application will not verify Java API types.

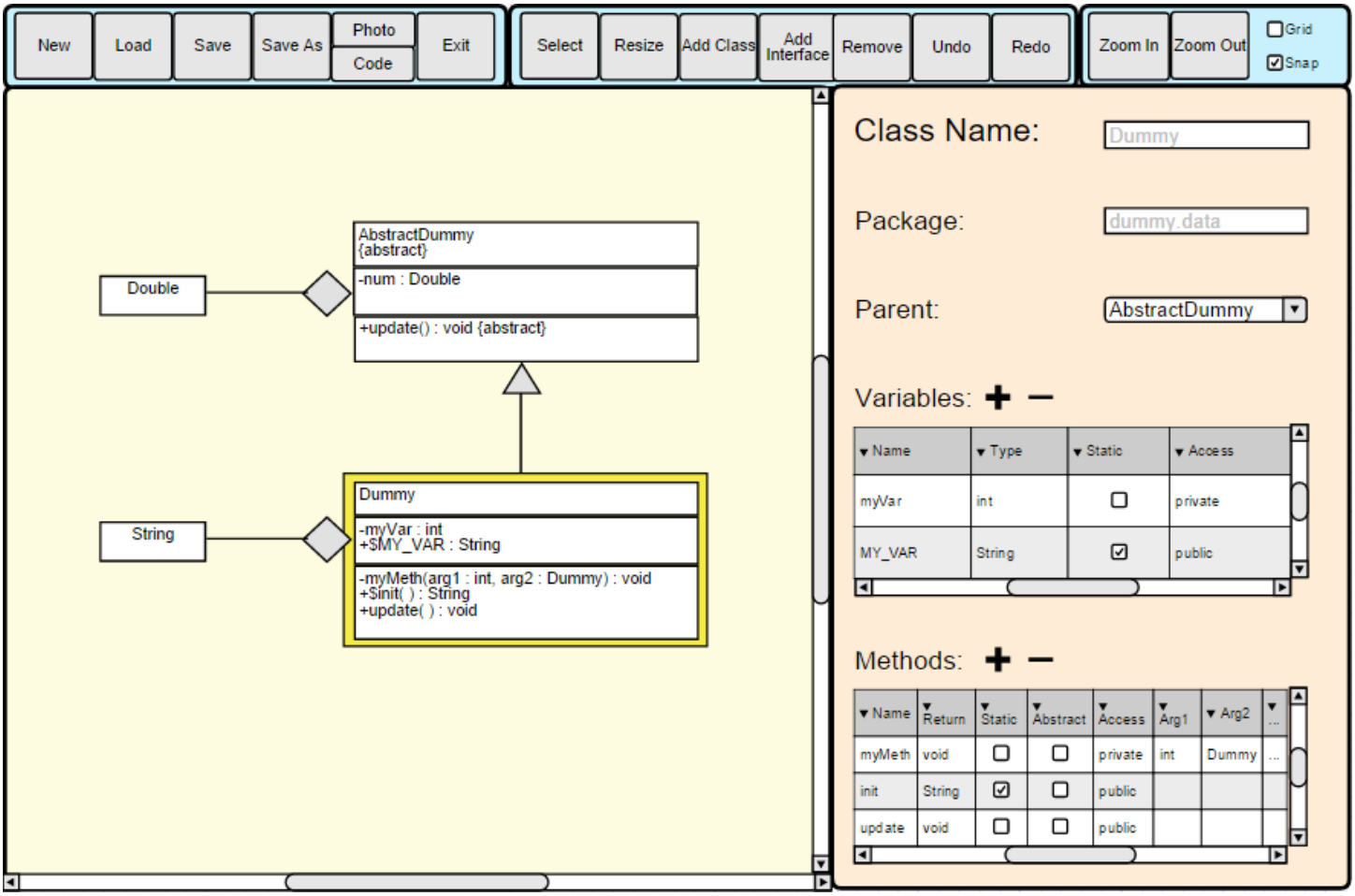
2.6 Apportioning of the Requirements

Note that the benchmark scheduling is specified in the table but this is subject to change depending upon progress made along the way. Slippage in ones schedule is always an issue and so periodic review of schedule may help.

3 Specific requirements

The jClass Designer user interface is to be carefully designed such that it accommodates all Use Cases specified in this document. The UI designer should aggregate all needed controls and then apportion them to their proper toolbars and other controls containers so as to make the application easy to use and attractive.

3.1 **External interfaces**



3.2 **Functions**

One of the important things to consider in our application is providing the appropriate feedback to the user while the application is running. Users need feedback to enjoy their experience. This is typically done with visual cues like dialog boxes or highlighting (like when components are selected).

3.3 **Performance requirements**

N/A

3.4 **Logical database requirements**

N/A

3.5 **Design constraints**

JavaFX will be used because it effectively leverages each system's available rendering technologies and provides platform independence for personal computers. Therefore the assumption is that the user will have a mouse pointing device and keyboard. We will not assume any other devices.

3.6 **Software system attributes**

As professionals, all members of this project must take this project seriously. We are dedicated to producing robust software that exceeds the expectations of our customers. In order to achieve this level of quality, we should build a product with the following properties in mind:

3.6.1 Reliability – The program should be carefully planned, constructed and tested such that it behaves flawlessly for the end user. Bugs, including rendering problems, are unacceptable. In order to minimize these problems, all software will be carefully designed using UML diagrams and a Design to Test approach should be used for the Implementation Stage.

3.6.2 Availability –Customers may download and install the application for free.

3.6.3 Security – All security mechanisms will be addressed by future revisions

3.6.4 Extensibility – It is important that more features can be added to the application, so file formats for should be carefully considered such that the game can be easily extended.

1.6.5 Portability – To start with, the app will target desktop Java applications. Future ports may be as a Web app.

3.6.6 Maintainability – Update mechanisms will be addressed by future revisions.

3.7 **Organizing the specific requirements**

Note that the application is simple enough that we need not worry about using an alternative arrangement of the content of this document. The specific requirements for this application already fit neatly into the sections listed in the IEEE's recommended SRS format.

3.8 Additional comments

It is important to keep in mind that the UI designers, map creators, and sound designers should make updates to the game themes and content as need to make something that looks great. It will be to their discretion to design all the interface controls in an effective, interactive style.

4 Supporting Information

Note that this document should serve as a reference for the designers and coders in the future stages of the development process, so we'll provide a table of contents to help quickly find important sections.

4.1 Table of contents

4.2 **Appendixes**

N/A