BlueJ Text Editor Design

by
Holmes, Benjamin
Jain, Ayushi
Magee, Kylyn

Submitted in partial fulfillment
of the requirements for
CS 7140 Course

Department of Computer Science
Wright State University
24 October 2013
This document is written to concretely establish a design for the developers, based on the concepts put forth in the requirements and specifications documents. It will do so by using ideas for the design patterns and implementing the principles of Design by Contract. This document will further eliminate any ambiguity that might have been left unattended in the previous documents. This document focuses solely on the design specification of the text editor portion of the BlueJ application. It will thus discuss the features of the text editor, its GUI, the design patterns and principles and provide a detailed description of the extension idea presented in the previous document: package maintenance. It will also discuss the mathematical and logical components and describe the specifications for buffers, cursors, view-windows, cut-paste-containers, keyboard and mouse input methods, undo/redo, etc.
# Table of Contents

1. Introduction .................................................................................................................................. 5

2. Operations .................................................................................................................................... 5
   2.1 Window ................................................................................................................................... 5
   2.2 Buttons ................................................................................................................................... 5
   2.3 Cut .......................................................................................................................................... 6
   2.4 Copy Copy is implemented similarly with the following steps: .............................................. 6
   2.5 Paste ...................................................................................................................................... 7
   2.6 Undo/redo ................................................................................................................................. 7

3. User Interface ................................................................................................................................ 7
   3.1 Views ...................................................................................................................................... 8
      3.1.1 Project manager (A) .......................................................................................................... 8
      3.1.2 Source code editor (B) ...................................................................................................... 8
      3.1.3 Terminal (C) ..................................................................................................................... 10
      3.1.4 Javadoc display (D) .......................................................................................................... 10
      3.1.5 Code pad (E) ..................................................................................................................... 10
   3.2 Panels ...................................................................................................................................... 11
      3.2.1 Menu bar (F) ..................................................................................................................... 11
      3.2.2 Quick reference buttons (G) ............................................................................................. 14
      3.2.3 Program testing features (H) ............................................................................................ 14

4. Plug-in .......................................................................................................................................... 14
   4.1 New Package ........................................................................................................................... 14
      4.1.1 Prompt ................................................................................................................................ 14
      4.1.2 Folder creation .................................................................................................................... 14
      4.1.3 Contents ............................................................................................................................. 14
   4.2 Move To ................................................................................................................................... 15
      4.2.1 Copying folder(s)/file(s) ................................................................................................... 15
      4.2.2 Updating source files ........................................................................................................ 15

5. Code Reduction ............................................................................................................................ 15

6. Maintainability ............................................................................................................................... 16

7. Conclusion .................................................................................................................................... 16

References ......................................................................................................................................... 17
**Table of Figures**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>User Interface with Default Views</td>
<td>7</td>
</tr>
<tr>
<td>3.2</td>
<td>User Interface with Alternate Views</td>
<td>8</td>
</tr>
<tr>
<td>3.3</td>
<td>Example of scope highlighting [5]</td>
<td>9</td>
</tr>
<tr>
<td>3.4</td>
<td>Project Menu</td>
<td>11</td>
</tr>
<tr>
<td>3.5</td>
<td>Project &gt; New... menu</td>
<td>11</td>
</tr>
<tr>
<td>3.6</td>
<td>Project &gt; Open... menu</td>
<td>12</td>
</tr>
<tr>
<td>3.7</td>
<td>Class menu</td>
<td>12</td>
</tr>
<tr>
<td>3.8</td>
<td>Edit Menu</td>
<td>12</td>
</tr>
<tr>
<td>3.9</td>
<td>Tools Menu</td>
<td>13</td>
</tr>
<tr>
<td>3.10</td>
<td>View menu</td>
<td>13</td>
</tr>
<tr>
<td>3.11</td>
<td>Help menu</td>
<td>14</td>
</tr>
</tbody>
</table>
1. Introduction
Having read the previous, Requirements and Specification, documents for the text editor of BlueJ, one may now want to look into an in-depth description of the design used for implementing the text editor, its properties and functionalities.

2. Operations
Discussed in this section, are details pertaining to the operations that the text editor is expected to perform. Many of the operations that we want to include are made relatively simple with the use of Python.

For this example, we will use python gtk [1], which, in addition to providing a simple way to implement GUIs, is multi-platform, and updated regularly. It’s also freeware. This simplifies (from a user’s perspective), the creation and maintenance of actions relating to the text editor window.

2.1 Window
The basic structure of implementing most of the operations needed for our text editor comes from the creation of a window, accomplished in gtk simply with the following snippet of code. The snippet describes the constructor necessary to build a text editor window.

```python
class TextWindow(Gtk.Window):
    __gtype_name__ = "TextWindow"

    def __init__(self):
        super(TextWindow, self).__init__()
        self.connect('delete-event', Gtk.main_quit)

        self.vbox = Gtk.VBox(False, 8)
        for x in range(4):
            self._build_entry()
```

2.2 Buttons
Buttons can then be added which implement the functions we’re looking for, i.e. cut, copy, paste, undo, redo.

The creation of the buttons themselves is simple in gtk python. A button is created with an appropriate label, and when the button is clicked, its state is set to clicked.

```python
cut_button = Gtk.Button(label='Cut')
cut_button.connect('clicked', self.on_cut_clicked)
self.vbox.pack_start(cut_button, False, False, 0)

copy_button = Gtk.Button(label='Copy')
```
copy_button.connect('clicked', self.on_copy_clicked)
self.vbox.pack_start(copy_button, False, False, 0)

paste_button = Gtk.Button(label='Paste')
paste_button.connect('clicked', self.on_paste_clicked)
self.vbox.pack_start(paste_button, False, False, 0)

undo_button = Gtk.Button(label='Undo')
undo_button.connect('clicked', self.on_undo_clicked)
self.vbox.pack_start(undo_button, False, False, 0)

redo_button = Gtk.Button(label='Redo')
redo_button.connect('clicked', self.on_redo_clicked)
self.vbox.pack_start(redo_button, False, False, 0)

Then add those buttons to the vbox, and set up the clipboard.

self.add(self.vbox)
self.show_all()
self.clipboard = Gtk.Clipboard.get(Gdk.SELECTION_CLIPBOARD)

2.3 Cut
As an example, a snippet of code describing the cut operation is presented below.

def on_cut_clicked(self, widget):
    # First, we see what the selected text bounds are
    selected = self.focus.get_selection_bounds()

    # if the selection isn't an empty tuple,
    # save the selection as the variable chars
    # and copy it to the clipboard
    # then delete the selection
    if selected:
        chars = self.focus.get_chars(*selected)
        # following line is for testing
        # print "Copying '%s' from: %s" % (chars, self.focus)
        self.clipboard.set_text(chars, -1)
        # following line is for testing
        # print "Deleting text selection: (bounds[0], bounds[1])"
        self.focus.delete_text(bounds[0], bounds[1])
    else:
        print "Please select text for cutting"

2.4 Copy
Copy is implemented similarly with the following steps:
1. Check the range of the selection to ensure that some text has been selected.
2. Write the selected text, if there is any, to the clipboard.
3. Do nothing if there is no text selected.

2.5 Paste
Paste retrieves the cursor position with `pos = self.focus.get_position()` then inserts the elements of the clipboard at that position. If there is nothing in the clipboard, do nothing.

2.6 Undo/redo
Undo and Redo would require the use of a series of save states. This will be implemented with a list of insert and delete events - when text is entered, or one of the buttons is pressed, a list is kept of that insert or delete. An undo would simply locate the text, with position, that was inserted or deleted, and perform the opposite action. This event would be added to a second list, which would be the redo list. All of this will be handled by Java Swing libraries[2][3].

3. User Interface
The BlueJ application will use one window, divided into three main views and three panels [4]. The first of the views will be fixed as the project manager view. The other two views will be occupied by the source code editor and Javadoc display, and the terminal and code pad, respectively. The three panels will contain the menu bar, the quick reference buttons, and the program testing features, respectively (Figures 3.1 and 3.2).

![Figure 3.1: User Interface with Default Views](image)
3.1 Views

3.1.1 Project manager (A)
The project manager view is where users will create classes and define relationships between classes. They will be able to visualize and organize their projects here without opening a source file. Here the users are encouraged to build the skeleton for their program (i.e. design) before attempting the flesh it out (i.e. implementation). More details about this view will be provided in a separate document that focuses on the project manager specifications.

3.1.2 Source code editor (B)
The source code editor is where users will write and edit the source of the classes belonging to the project that is loaded into BlueJ. The editor will boast scope and syntax highlighting, as well as automatic file save. Additionally it will have tools such as auto-layout, insert method, block comment out, and find/replace. These features have been selected to be included in the source editor because of their usefulness to beginning programmers and their relative ease of use.

Scope highlighting
Scope highlighting will make it easy to see whether brackets are matched with a quick visual scan.
As shown in Figure 3.3, the background color for a particular scope begins with the opening bracket and ends with the closing bracket, which define the scope. Each nesting level will be highlighted with a different color.

Syntax highlighting
The editor will use syntax highlighting to differentiate certain Java keywords (access modifiers, decision structures, etc), primitive types, string literals, boolean literals, and comments.

Automatic file save
Source files will be saved automatically when the editor is closed and before code compilation.

Auto-layout
The auto-layout tool will be used to automatically correct inconsistent indentation in a source file. Each scope in a Java source code file should have a different level of indentation. In order for the indentation to be correct, each line within a scope must be at the same level of indentation. The auto-layout tool will use indentation to align each scope at the correct level of indentation. Additionally, the auto-layout tool will correct the size of the indentation itself to four spaces as specified by the Java Code Conventions [6]. If the source file already has correct indentation, a message will be displayed verifying that the indentation was already correct at the time this action was performed. Otherwise, all lines of source code in the file will be corrected.

Insert method
The insert method tool will insert a “template” method into the source code file, starting at the location of the cursor. By template, we refer to a method which has been written in advance and
will just be copied into the file. The method will be simple, but still a complete working example. It will have an access modifier, a non-void return type, and at least one parameter. This method will have an accompanying comment block that includes a brief description as well as parameter and return values. The insertion of this method will not cause compilation errors if the method is inserted at an appropriate place (e.g. at the class level).

**Block comment out**

The block comment out tool will comment out one or more lines of code in a source code file that has been loaded into the editor. When a user wants the compiler to ignore a line of code, but is not ready to delete the line, the user may prepend the line with two backslashes. This is an example of commenting out code. This tool will comment out all lines in a selected region. If the point and mark are on the same line, only that line will be commented out.

**Find/replace**

The find/replace tool will search a source code file for a given string. When the string is found, the text background of every matched string will be set to blue. The user will be able to navigate the matched strings. The text “in focus,” or currently selected, will have a yellow background color. The user will be able to change which matched string is in focus by navigating forward to the next matched string or backward to the previous matched string. If the user navigates forward from the last matched string, the find/replace tool will continue to the first matched string. If the user navigates backward from the first matched string, the tool will continue to the last matched string.

This tool will also allow the user to replace matched strings with a given replacement string. The replace function will replace the string in focus with the replacement string. There will also be a replace all function that will replace all matched strings with the replacement string.

3.1.3 Terminal (C)

The terminal will behave as a console that has been tasked specifically to run whatever project is loaded into BlueJ. It will be the standard in and out for programs run from BlueJ. The terminal will only accept input for a program running in BlueJ, and only if the program is listening to the terminal for input.

3.1.4 Javadoc display (D)

The Javadoc display will allow the user to see the HTML documentation generated for their class by Javadoc. This view will be read-only, but the user may copy and search the text in this view.

3.1.5 Code pad (E)

The code pad will let the user experiment with Java syntax, accepting both expressions and statements, and producing immediate results. The code pad will accept keyboard input terminated by a newline character. Input text will be black and output text will be green.
3.2 Panels

3.2.1 Menu bar (F)
The menus and menu items will follow the conventions set forth by other IDEs (such as Eclipse), but will be organized to suit beginning programmers. It is for this reason that the application will have 6 menus: Project, Class, Edit, Tools, View, Help.

**Project**
The Project menu will contain all operations which the user may perform on a project (Figure 3.4).

![Figure 3.4: Project Menu](image)

The Project menu will contain two submenus: New and Open. The New menu will contain all of the operations with which a user will create and add items to a project (Figure 3.5). The Open menu will contain all of the operations with which the user may load an existing project into BlueJ (Figure 3.6).

![Figure 3.5: Project > New... menu](image)
Class
The Class menu will contain all of the operations that may be performed on a single source code file (i.e. a Java class). The name of the class in focus will appear next to each item in this menu (Figure 3.7).

Edit
The Edit menu will contain some convenient tools with which the user may modify the contents of a source code file (Figure 3.8).
Tools
The Tools menu will contain tools that would be useful after a program has been written (Figure 3.9).

![Figure 3.9: Tools Menu](image)

View
The View menu will allow the user to toggle the visibility of the different windows and views that are available in BlueJ (Figure 3.10).

![Figure 3.10: View menu](image)

Help
The Help menu will contain items that provide the user with information about the installed BlueJ application, as well as links to online resources (Figure 3.11).
3.2.2 Quick reference buttons (G)
Panel G will contain quick reference buttons for operations available in the Project (New Class, Uses Relation, Inherits Relation) and and Tools (Compile All) menus. This panel will also include a drop list that will toggle the visibility of the Source Code and Documentation views.

3.2.3 Program testing features (H)
Panel H will contain features useful for program testing, a debugger status graphic and operation status text area [7], as well as a drop list that will toggle the visibility of the Terminal and Code Pad views.

4. Plug-in
The package management extension for BlueJ will support/enhance the following operations: new-package and move-to.

4.1 New Package

4.1.1 Prompt
The user can choose the New option from the Project menu and the Package option thereafter to create a new package. A prompt will then appear asking the user the package name that they would want to assign to it.

4.1.2 Folder creation
A new folder will then be created in the root folder which will be the project folder. If the names of the package should contain period(s), subfolders will be created delimited by the period(s). Example: if the user should input the name as basepackage.subpackage, a folder for basepackage will be created, containing the folder for the subpackage within it. If basepackage already exists, a new folder for the subpackage will be created within the basepackage folder.

4.1.3 Contents
Any package folder newly created (without specifically assigning a sub package to it using a period) will contain nothing in it initially.
4.2 Move To
The selected class/package can be moved to another desired package. The available packages will be made visible to the user through a browse window. This process shall entail the following steps: copying the folder(s)/file(s) to the provided destination, and updating the source code files to reflect change in project organization [8].

4.2.1 Copying folder(s)/file(s)
Copying the selected class/package to be moved, to its destination location. Once the copy operation completes successfully, the original copy shall be deleted from the source location. If, for any reason, the copy operation fails, whatever changes have been done, will be undone and the user will be informed about the operation failure.

4.2.2 Updating source files
Once the moving has been done, the import statements need to be changed in classes that import the moved class/package. If the moved class has been imported by any other class, an exact match for the moved class’ name will be looked up for in the importing class’ code. This implies that the name of the moved class will be captured and changed to the new destination packagename.classname, only if it is written in the following manner: import originalpackage.MovedClassName;

Limitation
This operation will not update imports that are written like the following statement: import originalpackage.*; This limitation is to encourage the beginner users to import particular classes that they will be using in their code and not the entire package as it is not a good programming practice to import unnecessary classes and packages.

Subpackages
The same protocol shall be followed for subpackages as well. On successful completion of moving the files, the user will be informed accordingly.

5. Code Reduction
Code reduction is essential in order to reduce the complexity of the program and to promote better maintainability [9]. The goal of 5% size reduction in code is aimed to be achieved during the implementation phase. This is because the design is ought to be written in a very high level language and it is truly the implementation which is executable. Once the executables are created, and they work just as their functionalities are described, that is when our attention can be diverted to code reduction. Some of the many techniques that can be applied for code reduction include:
Refactoring out duplicate code: A very common error observed is unnecessary duplication of code. This code can easily be deleted and the relevant references be made to the sole copy of the earlier duplicated code.
Eliminating dead code and unused variables: Allocating memory for variables that are just not used anywhere in the entire program can easily be called off and so can be parts of code which never really execute. Such dead code can also result due to some loping conditions in which the
controls never really reaches some parts of the code but the execution is still perfect. Such code can easily be eliminated without hampering the execution.

6. Maintainability
A dedicated maintenance team shall be looking into the maintenance aspects of the code. Refactoring will reduce the size of the code for better maintenance purposes [10]. The common misconception of maintenance being more of bug fixing shall be eliminated as this process will also (majorly) include non-corrective actions which in reality will be functionality enhancements of the overall text editor. Suggestions for these functionality enhancements shall be obtained from users who would then be currently working on projects in BlueJ and can give their feedback through BlueJ websites.

7. Conclusion
The construction of our text editor, and implementation of the GUI and the extension, will be greatly facilitated by the use of the Python high level language. We will construct a window using the gtk library in Python, and use this syntax to add in the necessary operations - the creation of a GUI will also be facilitated with this library.
References

Direct

Indirect
Appendix

A. Journals

A.1 Ayushi
10/18: Kylyn, Ben and I met and discussed the document’s expectations. Created the google doc and the code repository. Divided responsibilities: Kylyn will do section 2.1-2.2, Ben will handle section 2.5 and I will deal with section 2.4. (0.5 hours)
10/20: Read and tried comprehending parts of the BlueJ code that I was supposed to. (2 hours)
10/23: Kylyn and I met and outlined the entire document. We also designed the details of our plugin. (2 hours)
10/24: Edited the plug-in section of the document. (2 hours)
10/24: Wrote the Maintenance and Code Reduction Sections. (2 hours)
10/24: Kylyn and I together merged Ben’s content with this document and did the finishing changes for the document like adding the table of contents, list of figures, etc. (1 hour)

A.2 Ben
10/18 - met with group, divided up tasks for tech document section. Took operations, since I worked on those last time (30 min)
10/21 - downloaded bluej source code, analyzed (2 hours)
10/21 - looked for GUI examples in Python. Found a few minimal text editors, which I analyzed. Re- Downloaded bluej, analyzed source code (2 hours)
10/22 - found gtk library for Python, read tutorials, found how to design/implement buttons (1 hour)
10/24 - wrote and tested more code for GUI operations buttons, wrote a few additional things for finished document (2 hours)

A.3 Kylyn
10/18 -- Team Planning: Discussed document expectations, developed plan for timely completion, and divided responsibilities. (0.5 hours)
10/23 -- Writing: Ayushi and I met to work on the design for the package management extension. We also discussed expectations for design by contract and functionality of redesigned GUI. (2 hours)
10/24 -- Writing and Editing: Ayushi and I merged Ben’s contributions into the documents. I added some content to the Operations and made some modifications to Ben’s content. I also “smoothed” some sentences and added consistent formatting throughout the document. (2.5 hours)