Assignment 1.1 - Extending Your Chess Library

Overview

This week, we will be focusing on refactoring your code from last week, adding good documentation, and also extending your library by adding two custom pieces and the controller component of MVC. Refactoring your library code and adding good documentation now - and throughout the assignment 1.X - will allow for easier and more efficient development in the coming weeks. By the end of the week, you should have a clean, easy-to-understand, and extensible library.

For this assignment, you are required to use either Eclipse or IntelliJ IDEA. Both are free and have powerful refactoring tools available.

Eclipse vs. IntelliJ IDEA
You are likely already familiar with Eclipse from earlier programming courses here at UIUC. A few staff prefer IntelliJ. If you've never tried it out, consider using it for this project.

Assignment Format
This course is likely very different previous courses you have taken, in that we typically **reuse** your code from the previous week for each assignment. As such, don't waste your time with messy code: focus on maintainability.

Read this entire page before beginning your assignment, and post on Piazza if anything is still unclear.

Part I: Refactoring & Polishing Test Suite

Agile Mantra
"Make it work. Make it right. Make it fast."

- Kent Beck

Read here and here for further discussion about this topic.

You spent Assignment 1.0 *making it work*, now you will:

- **Refactor** your code to *make it right*.
- *Make it faster* by using proper data structures.

Before you begin refactoring, consider your test suite from Assignment 1.0. Did your moderator, the TAs, or your peers from discussion section suggest ways to improve the coverage of your test suite? Yes, yes you should have written your tests *before* implementing your functionality last week, but if for whatever reason, your test suite could be more thorough, spend some time

enriching your test suite *before* you begin refactoring. Doing so will help you refactor more quickly, and to be more confident in the correctness of your refactorings.

If you have not already done so, consider using a code coverage tool such as EclEmma for Eclipse, or the integrated code coverage features in IntelliJ IDEA to quantify how thorough your test suite truly is.

You should refactor your code to eliminate any code smells (e.g. use communicative naming, decompose larger methods into smaller separate methods, etc), add missing tests, or other problems discussed in section.

Part II: Auto-generate Documentation

Next, use Doxygen to auto-generate documentation for your library. You can find pre-packaged binaries here, or run the following command on the EWS machines or other Linux distros to get the latest build of Doxygen. **I will assume this is run from the root directory of your project (i.e. from Assignment1.1/)**:

```
wget http://ftp.stack.nl/pub/users/dimitri/doxygen-1.8.2.linux.bin.tar.gz && tar xf doxygen-1.8.2.linux.bin.tar.gz && cp doxygen-1.8.2/bin/doxygen ./ && rm doxygen-1.8.2.linux.bin.tar.gz && rm -rf doxygen-1.8.2
```

Then, to automatically configure and generate documentation for your project, simply run the following:

1. Run **chmod a+rx doxygen && ./doxygen -g**
2. Modify *Doxyfile* line 688, and change *RECURSIVE* from *NO* to *YES*
3. Run **./doxygen Doxyfile**

If you have followed the instructions properly, your project directory should now contain autogenerated HTML & latex found under *html*and *latex*. Take a look at *html/index.html* in a browser to check it out!

This part should be relatively straightforward, and is intended to encourage you to expand on the documentation of your public methods and classes. Imagine you are handing this library to another developer, and that the PDF or HTML generated by Doxygen will be this programmer's first contact with your library. Are there any thinly documented areas of the code? It should be more obvious using Doxygen which areas could use further explanation. If you do see these areas, expand on your documentation, and run Doxygen again to regenerate your documentation.

*Please **do not** check your doxygen binary or generated documentation into SVN (the _latex*, *html*, or *doxygen* files). This uses up a tremendous amount of space and resources in SVN, and will result in a deduction from **Code Submission** on the rubric. **However, please ensure that *Doxyfile*, or whatever configuration file you use to generate your documentation, is**

committed to SVN. Without it, we cannot give you credit for the Doxygen-related requirements.

Part III: Two Custom Chess Pieces

If your code is properly refactored, this part should be a breeze. Your task is to create two custom chess pieces. All the usual chess rules apply to your custom pieces, for example, they are not allowed to move outside the board. For simplicity, your custom pieces do not have to implement special moves, just as we ignored castling for king and rooks. You may find this article from wikipedia useful, but you are free to create your own movement.

Part IV: Static Graphical User Interface

Your task for this part is to implement a **STATIC** GUI. By static, we mean that your GUI should have **ZERO USER INTERACTION**. The only thing required for this week is to display the initial configuration of a chess board. **Do not waste your time implementing chess moves, you may even lose points if you do**. The point of this restriction is to give you a clearer understanding of the MVC architecture, by strictly separating the Model, View, and Control components. You implemented the chess model last week, this week's focus is the view. Simply display a chess board with the normal set of chess pieces in their initial positions. Keep in mind, however, this does not mean you can simply display a static image in your JFrame.

GUI Builder Warning!

If you decide to use the UI builder in Netbeans or IntelliJ for this assignment, be very careful.

Although the UI builder quickly generates Java GUIs, it produces very ugly code that will not meet the requirements for this assignment without modification. Most noticeably, it tightly couples the view with the controller, so significant refactoring of the autogenerated code will be required. In the staff's opinion, it is *more work* to generate a UI automatically and refactor it properly than to build one from scratch.

Hand written code

```
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
```

Hand written code

```
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.UIManager;


public class GUIExample implements ActionListener{

    public GUIExample(){
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch(Exception e) {
            //silently ignore
        }
        JFrame window = new JFrame("Basic Application Example");
        window.setSize(500, 500);
        JPanel myPanel = initializePanel();
        initializeButton(myPanel);
        setUpMenu(window);
        window.setContentPane(myPanel);
        window.setVisible(true);
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    private void initializeButton(JPanel myPanel) {
        JButton button = new JButton("Click me");
        button.addActionListener(this);
        myPanel.add(button, BorderLayout.SOUTH);
    }
```

Hand written code

```
    private JPanel initializePanel() {
        JPanel myPanel = new JPanel();
        myPanel.setPreferredSize(new Dimension(500,500));
        myPanel.setLayout(new BorderLayout());
        return myPanel;
    }

    private void setUpMenu(JFrame window) {
        JMenuBar menubar = new JMenuBar();
        JMenu file = new JMenu("File");
        JMenuItem open = new JMenuItem("Open");
        open.addActionListener(this);
        file.add(open);
        menubar.add(file);
        window.setJMenuBar(menubar);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(null,
                    "I was clicked by "+e.getActionCommand(),
                    "Title here", JOptionPane.INFORMATION_MESSAGE);
    }

    public static void main(String[] args) {
        new GUIExample();
    }
}
```

Netbeans autogenerated code

```
/*
 * GUIExampleView.java
 */
```

Hand written code

```
package guiexample;

import org.jdesktop.application.SingleFrameApplication;
import org.jdesktop.application.FrameView;
import javax.swing.JOptionPane;

/**
 * The application's main frame.
 */
public class GUIExampleView extends FrameView {

    public GUIExampleView(SingleFrameApplication app) {
        super(app);

        initComponents();


    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        mainPanel = new javax.swing.JPanel();
        menuBar = new javax.swing.JMenuBar();
        javax.swing.JMenu fileMenu = new javax.swing.JMenu();
        javax.swing.JMenuItem exitMenuItem = new javax.swing.JMenuItem();
        statusPanel = new javax.swing.JPanel();
        javax.swing.JSeparator statusPanelSeparator = new javax.swing.JSeparator();
        statusMessageLabel = new javax.swing.JLabel();
```

Hand written code

```
        statusAnimationLabel = new javax.swing.JLabel();
        jButton1 = new javax.swing.JButton();

        mainPanel.setName("mainPanel"); // NOI18N

        javax.swing.GroupLayout mainPanelLayout = new javax.swing.GroupLayout(mainPanel);
        mainPanel.setLayout(mainPanelLayout);
        mainPanelLayout.setHorizontalGroup(
            mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
            .addGap(0, 415, Short.MAX_VALUE)
        );
        mainPanelLayout.setVerticalGroup(
            mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
            .addGap(0, 222, Short.MAX_VALUE)
        );

        menuBar.setName("menuBar"); // NOI18N

        org.jdesktop.application.ResourceMap resourceMap =
            org.jdesktop.application.Application.getInstance(guiexample.GUIExampleApp.cla
            .getContext().getResourceMap(GUIExampleView.class);
        fileMenu.setText(resourceMap.getString("fileMenu.text")); // NOI18N
        fileMenu.setName("fileMenu"); // NOI18N

        exitMenuItem.setText(resourceMap.getString("exitMenuItem.text")); // NOI18N
        exitMenuItem.setName("exitMenuItem"); // NOI18N
        exitMenuItem.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                exitMenuItemActionPerformed(evt);
            }
        });
        fileMenu.add(exitMenuItem);

        menuBar.add(fileMenu);

        statusPanel.setName("statusPanel"); // NOI18N
```

Hand written code

```
statusPanelSeparator.setName("statusPanelSeparator"); // NOI18N

statusMessageLabel.setName("statusMessageLabel"); // NOI18N

statusAnimationLabel.setHorizontalAlignment(javax.swing.SwingConstants.LEFT);
statusAnimationLabel.setName("statusAnimationLabel"); // NOI18N

jButton1.setText(resourceMap.getString("jButton1.text")); // NOI18N
jButton1.setName("jButton1"); // NOI18N
jButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton1ActionPerformed(evt);
    }
});

javax.swing.GroupLayout statusPanelLayout = new javax.swing.GroupLayout(statusPan
statusPanel.setLayout(statusPanelLayout);
statusPanelLayout.setHorizontalGroup(
    statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
              statusPanelLayout.createSequentialGroup()
    .addGroup(statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Align
    .addGroup(statusPanelLayout.createSequentialGroup()
    .addContainerGap()
    .addComponent(statusMessageLabel)
    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 387, Sho
    .addComponent(statusAnimationLabel))
    .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE,
                  399, javax.swing.GroupLayout.PREFERRED_SIZE))
    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
    .addComponent(statusPanelSeparator))
);
statusPanelLayout.setVerticalGroup(
    statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
    .addGroup(statusPanelLayout.createSequentialGroup()
```

Hand written code

```
            .addContainerGap()
            .addGroup(statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Align
            .addComponent(statusPanelSeparator, javax.swing.GroupLayout.PREFERRED_SIZE,
                        2, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(jButton1))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
                        javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addGroup(statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Align
            .addComponent(statusMessageLabel)
            .addComponent(statusAnimationLabel))
            .addGap(3, 3, 3))
        );

        setComponent(mainPanel);
        setMenuBar(menuBar);
        setStatusBar(statusPanel);
    }// </editor-fold>

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        JOptionPane.showMessageDialog(null, "I was clicked by "+evt.getActionCommand(),
                                    "Title", JOptionPane.INFORMATION_MESSAGE);
    }

    private void exitMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
       jButton1ActionPerformed(evt);
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JPanel mainPanel;
    private javax.swing.JMenuBar menuBar;
    private javax.swing.JLabel statusAnimationLabel;
    private javax.swing.JLabel statusMessageLabel;
    private javax.swing.JPanel statusPanel;
    // End of variables declaration
```

Hand written code

```
}
```

Getting Started

GUI programming can be a potentially daunting experience. Below are some external resources to help get you going. Please start early and come to office hours or ask questions on Piazza if you are confused. These resources cover more than what you need for implementing a static GUI, but keep in mind **NO USER INTERACTION** this week.

- MVC Wikipedia Article - The Wikipedia article on Model-view-controller architecture.
- Sun JAVA GUI Tutorial - Sun's tutorial on Java GUI programming. This is a great resource since it is straight from the source.
- Model/View/Controller GUI - An introduction to the model view controller design scheme. Includes a Java GUI example. Especially relevant is section 3.1.2.0.1 Warning.
- Crash Course in Java GUI - An introduction to Java GUI programming. The links on this site are actually powerpoint presentations.
- Java GUI Examples - A number of examples using JFrame and JButton.

Part V: Manual Test Plan

GUI testing is difficult, especially just with unit tests. This week, in order to test your GUI, write a test plan including screenshots and specific steps for a human tester to follow - what a tester should do and what he/she should observe. Since your GUI this week is static, the test script should be very simple. You will be building on this test plan in the coming weeks.

Need help?

First, ask questions on Piazza. If you have a question, there is a pretty good chance someone else has the some one and an even better chance that someone else in the class or one of the TAs will be able to answer it for you. If you are still having a problem, email your moderator or one of the TAs to get advice. Remember, its best to ask questions early on so they have time to be answered. Don't wait until the last second to get started then realize that you are confused. In general, we are flexible with interpretations of the assignment, **as long as it does not trivialize any component of the assignment**.

Submission

This assignment is due at the **beginning of your discussion section the week of September 25, 2017**. Please be sure to submit in SVN, and ask your moderator or TA before the deadline if you have any questions. Also, make sure to place your work in a folder called **Assignment1.1**, matching this spelling exactly.

For example, you should be able to bring up https://subversion.ews.illinois.edu/svn/fa17-cs242/NETID/Assignment1.1/ in a web browser and see all of your code (if you replace NETID with your actual netid)

Objectives

- Clean up any problems in your code for Assignment 1.0, expand your test suite if necessary, and fix any algorithmic shortcomings
- Auto-generate documentation for your library
- Two custom chess pieces and static user interface using Observer Pattern.

Resources

- Design Patterns
- Design patterns
- Model-View-Controller architecture
- Observer Pattern
- Refactoring
- Refactoring
- Refactoring in Eclipse
- Refactoring in InteliJ

Grading

We will bias clarity over cleverness in evaluating your code. You should adopt a *"teaching"* mindset, always asking yourself, *"How would I change what I have written to make it as easy as possible for someone else to understand my code without my direct assistance?"*

Refer to the standard Grading rubric if any portion of this rubric is unclear. _ Note that the standard rubric assumes a scale of 0-2 for each category_.

Category	Weight	Scoring	Notes
Basic Preparation	2	0-1	Ready to go at the start of section
Cleverness	2	0-2	The hardest points on the rubric
Code Submission	4	0-2	Submitted correct content on time and to the correct location in the repository
Decomposition	4	0-2	Project is adequately decomposed to different classes and methods
Documentation	4	0-2	Comments for each class and each function are clear and are following style guides
Effort	2	0-2	Perform considerable amount of work
Naming	2	0-2	Variable names and method names are readable and are following Java conventions
Overall Design	4	0-2	Have nice approaches and structures in overall
Participation	5	0-2.5	Interact with the group 2 times (ask a question, make a comment, help answer a question, etc.)
Presentation	4	0-2	Present the code clearly
Requirements - Custom Chess Pieces	5	0-2.5	2 points - Custom Chess Pieces are a simple combination of two standard chess pieces. Any piece which is in thislinkwould constitute 2 points. 2.5 points - Custom Chess Pieces are unique and the algorithm used to describe its movement is complex and

Category	Weight	Scoring	Notes
			different from the movements of standard chess pieces.
Requirements - Static User Interface	5	0-2.5	2 points - The UI consists of a neat layout which accurately represents a chess board. The UI consists of pictures for each of the chess pieces. 2.5 points - The UI is extremely appealing and shows that the student has put inconsiderableeffort in the design of the layout. The UI consists of clickable buttons for each tile and basic interactions are enabled in the UI.
Requirements - Doxygen Generation	4	0-2	2 points - Generate and submit a doxygen file
Requirements - Refactoring	5	0-2.5	2 points - There is not much duplicate/redundant code and student has put effort to refactor code from last week. There is minor but not significant scope for improvement. 2.5 points - Every piece of code is functional and overall design of the project is so well designed that itcannotbe refactored/improved any further.
Testing	5	0-2.5	Unit tests written for all new code & expanded last week's tests if necessary 2 points - 80% of the new code is tested and the tests cover edge cases for each function. 2.5 points - 95% of the new code has been tested and every

Category	Weight	Scoring	Notes
			possible edge cases has been tested for.
Testing (Script)	4	0-2	2 points - Make a complete manual test plan for GUI
Total	61		