IT Team Project Code Listing

Team J

1 Main.java

```
import javax.swing.UIManager;

public class Main {

   public static void main(String[] args) {

      // this code is just so it looks the same on a mac as windows
      try {
         UIManager.setLookAndFeel(
            UIManager.getCrossPlatformLookAndFeelClassName()
         );
      } catch (Exception e) {
         e.printStackTrace();
      }
      new ProgramController();
   }

}
```

2 Module.java

```
/**
 * Simple class to store details about a module.
 */
public class Module {

   //========================================================================
    // Properties
    //========================================================================

   /** The unique identifying code for the module. */
   private String code;

   /** The name of the module. */
   private String name;

   /** The number of students taking the module. */
   private int size;

   //========================================================================
    // Constructor
    //========================================================================
```

```
22
    /**
24  * Instantiate a module from a line of ModulesIn.txt.
    */
26  public Module(String code, int size, String name)
    {
28      this.code = code;
        this.size = size;
30      this.name = name;
    }
32
    //========================================================================
34   // Get methods
     //========================================================================
36
    /**
38   * Get the subject and year code for the module.
     * @return the 3 digit code.
40   */
    public String getSubjectYear()
42  {
        return code.substring(0,3);
44  }

46  /**
     * Get the full identifying code for the module.
48   * @return the module's unique identifying code.
     */
50  public String getCode()
    {
52      return code;
    }
54
    /**
56   * Get the number of people in taking the module.
     * @return the size of the module.
58   */
    public int getSize()
60  {
        return size;
62  }

64  /**
     * Get the name of the module. This is only used for file output.
66   * @return the full name of the module.
     */
68  public String getName()
    {
70      return name;
    }
72 }
```

3 Slot.java

```java
/**
 * Simple class representing a slot in the timetable.
 */
class Slot
{
   /** The time of the slot. */
   private String time;

   /** The room name. */
   private String name;

   /** The capacity of the slot. */
   private int size;

   /**
    * Get the time of the slot.
    * @return the time.
    */
   public String getTime()
   {
      return time;
   }

   /**
    * Get the name of the slot.
    * @return the name.
    */
   public String getName()
   {
      return name;
   }

   /**
    * Get the capacity of the slot.
    * @return the capacity.
    */
   public int getSize()
   {
      return size;
   }

   /**
    * Instantiate a slot from a given time name and capacity.
    * @param time the time.
    * @param name the name.
    * @param size the capacity.
    */
   public Slot(String time, String name, int size)
   {
      this.time = time;
      this.name = name;
      this.size = size;
   }
}
```

4 ProgramModel.java

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map.Entry;
import java.util.Objects;
import java.util.Arrays;

/**
 * The model for the program; stores the timetable, and contains methods
 * to retrieve information about the it, and to update it as required.
 */
public class ProgramModel {

    //============================================================================
     // Properties
     //============================================================================

    /** Number of different class times. */
    private final static int ROWS = 10;

    /** Number of different rooms. */
    private final static int COLS = 8;

    /** Array of class times. */
    private final String[] times = new String[] {
        "MonAM",
        "MonPM",
        "TueAM",
        "TuePM",
        "WedAM",
        "WedPM",
        "ThuAM",
        "ThuPM",
        "FriAM",
        "FriPM"
    };

    /** Array of room names. */
    private final String[] roomNames = new String[] {"A","B","C","D","E","F","G","H"};

    /** Array of room sizes. */
    private final int[] roomSizes = new int[] {100,100,60,60,60,30,30,30};

    /** 2D array of slots in the timetable. */
    private Slot[][] slots;

    /** {@link HashMap} enabling looking up the module in a given slot. */
```

```
    private HashMap<Slot, Module> schedule = new HashMap<Slot, Module>();

    /** Array of all the module that need scheduling. */
    private Module[] modules;

    //============================================================================
     // Get methods
     //============================================================================

    /**
     * Get method for the slots of the timetable.
     * @return 2D array of slots.
     */
    public Slot[][] getSlots()
    {
       return slots;
    }

    /**
     * Get method for the modules to be scheduled.
     * @return array of modules.
     */
    public Module[] getModules()
    {
       return modules;
    }

    /**
     * Get the slots into which a module has been placed.
     * @return array of filled slots.
     */
    public Slot[] getFilledSlots()
    {
       return schedule.keySet().toArray(new Slot[0]);
    }

    //============================================================================
     // Constructor and helper methods
     //============================================================================

    /**
     * Instantiate the program model by creating the slots and the modules.
     */
    public ProgramModel()
    {
       createSlots();
       createModules();
    }

    /**
     * Create the 2D array of slots representing the positions in the timetable.
     */
    private void createSlots()
    {
```

```
        // Create slots 2D array.
108     slots = new Slot[ROWS][COLS];

110     // Populate slots array with new slot objects.
        for (int i = 0; i < ROWS; i++)
112         for (int j = 0; j < COLS; j++)
                slots[i][j] = new Slot(times[i], roomNames[j], roomSizes[j]);
114 }

116 /**
     * Create the array of modules from the ModulesIn.txt file.
118  */
    private void createModules()
120 {
        // get the lines of the file and create module array of same length
122     String[] lines = getFileLines();
        modules = new Module[lines.length];
124
        // loop over the lines
126     for(int i = 0; i < lines.length; i++)
        {
128         // extract the fields from each and create a module object
            String[] ln = lines[i].split(" ");
130         modules[i] = new Module(ln[0], Integer.parseInt(ln[4]), ln[1]);

132         // schedule the module if time and room have been provided
            if (!ln[2].equals("?????"))
134             addModuleToSlot(
                    modules[i],
136                 slots[
                        Arrays.asList(times).indexOf(ln[2])
138                 ][
                        Arrays.asList(roomNames).indexOf(ln[3])
140                 ]
                );
142     }
    }
144

    /**
146  * Read the lines of the input file and return an array of strings,
     * one for each line.
148  * @return array of strings representing the lines in the file.
     */
150 private String[] getFileLines() {

152     // create a buffered reader for the input file
        BufferedReader in = null;
154     try {
            in = new BufferedReader(new FileReader("ModulesIn.txt"));
156     } catch (FileNotFoundException e) {
            e.printStackTrace();
158     }

160     // loop through the lines of the file and add each to a list
```

```
        String str;
        List<String> list = new ArrayList<String>();
        try {
            while((str = in.readLine()) != null)
                list.add(str);
        } catch (IOException e) {
            e.printStackTrace();
        }

        // return the list as an array
        return list.toArray(new String[0]);
    }

    //============================================================================
     // Program methods
     //============================================================================

    /**
     * Add a module to a given slot in the schedule.
     * @param module the module to schedule.
     * @param slot the slot to put it in.
     */
    public void addModuleToSlot(Module module, Slot slot)
    {
        // add module to slot in schedule
        schedule.put(slot, module);
    }

    /**
     * Obtain an array of all slots into which a given module
     * can be placed, according the rules of the scenario.
     * @param module the module to be placed.
     * @return array of valid slots.
     */
    public Slot[] validSlotsForModule(Module module)
    {
        // we will be returning an array of slots of indeterminate length, so
        // use array list
        ArrayList<Slot> s = new ArrayList<Slot>();

        // for each slot
        for (int i = 0; i < ROWS; i++)
            for (int j = 0; j < COLS; j++)
                // check if module fits, and if it does, add to our array list
                if (moduleFitsInSlot(module, slots[i][j]))
                    s.add(slots[i][j]);
        // convert array list to regular array on return
        return s.toArray(new Slot[0]);
    }

    /**
     * Check whether a given module may be legally placed into a given
     * slot.
     * @param module the module to be placed.
```

```
     * @param slot the slot to test.
     * @return true if the module may be placed in the slot, false otherwise.
     */
    public boolean moduleFitsInSlot(Module module, Slot slot)
    {
        // is a module already scheduled for the slot? If so return false
        if (schedule.get(slot) != null)
            return false;
        // does the slot have enough seats for the module? If not, return false
        if (module.getSize() > slot.getSize())
            return false;
        // is there already a class for that year at this time? If so, return false:
        // to check this condition, first get the row of the module in the slots array
        int t = Arrays.asList(times).indexOf(slot.getTime());
        // loop over all slots in that row (at that time)
        for (int i = 0; i < COLS; i++)
        {
            // get the module in the slot
            Module m = schedule.get(slots[t][i]);
            // if there is a module scheduled, and it is the same subject and year
            // return false
            if (m != null)
                if (m.getSubjectYear().equals(module.getSubjectYear()))
                    return false;
        }
        // if none of above, return true
        return true;
    }

    /**
     * Get the slot into which a given module has been placed,
     * or null if it has not been scheduled yet.
     * @param module the module for which to find the slot.
     * @return the slot in which the module is scheduled.
     */
    public Slot slotForModule(Module module)
    {
        // loop over the modules which have been scheduled
        for (Entry<Slot, Module> m : schedule.entrySet())
        {
            // if one is equal to module in question, return its key (slot)
              if (Objects.equals(module, m.getValue())) {
                  return m.getKey();
              }
         }
        // otherwise return null as module not scheduled
         return null;
    }

    /**
     * Get the module in a given slot.
     * @param slot the slot for which to get the module.
     * @return the module in the slot, or null if there is none.
     */
```

```
    public Module moduleInSlot(Slot slot)
270 {
       // look up slot in the schedule HashMap
272    return schedule.get(slot);
    }
274
    /**
276  * Get the string description of a module as it should appear
     * in the module view.
278  * @param module module for which to obtain description.
     * @return the string description.
280  */
    public String lineForModule(Module module)
282 {
       // if module not scheduled, slot description ends in question marks
284    String slotDescription = "  -  ?????  ?";
       // get the slot into which the module has been placed
286    Slot s = slotForModule(module);
       // if the module has been scheduled, replace question marks with info
288    if (s != null)
          slotDescription = "  -  " + s.getTime() + "  " + s.getName();
290    // return result
       return module.getCode() + "  " + module.getSize() + slotDescription;
292 }

294 //========================================================================
     // File Saving methods
296  //========================================================================

298 /**
     * Get the string description of a module as it should appear in the output
300  * text file.
     * @param module the module for which to obtain the description.
302  * @return the string description.
     */
304 private String outputLineForModule(Module module)
    {
306    // if module is not scheduled, slot description ends in question marks
       String slotDescription = "????? ?";
308    // get the slot into which the module has been placed
       Slot s = slotForModule(module);
310    // if the module has been scheduled, replace question marks with info
       if (s != null)
312       slotDescription = s.getTime() + " " + s.getName();
       // return result
314    return module.getCode() + " " + module.getName() + " "
          + slotDescription + " " + module.getSize();
316 }

318 /**
     * Write the module details to the output file.
320  */
    public void saveToFile()
322 {
```

```
        // the string that will hold all the modules' details
        String s = "";
        // loop through all modules and add their details
        for(int i = 0; i < modules.length; i++)
            s += outputLineForModule(modules[i]) + "\n";

        // make a Print Writer object
        PrintWriter writer = null;
        try {
            writer = new PrintWriter("ModulesOut.txt");
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }

        // write module details to file and close print writer
        writer.println(s);
        writer.close();
    }
}
```

5 TimetableView.java

```
import javax.swing.*;
import javax.swing.border.EmptyBorder;
import javax.swing.border.LineBorder;
import javax.swing.text.SimpleAttributeSet;
import javax.swing.text.StyleConstants;
import javax.swing.text.StyledDocument;

import java.util.*;
import java.awt.*;

/**
 * Class for the timetable component of the GUI. Contains methods to
 * update cells based on user input, and a button for each slot.
 */
@SuppressWarnings("serial")
class TimetableView extends JPanel
{
    //============================================================================
     // Properties
     //============================================================================

    /** HashMap allowing buttons to be looked up via slots. */
    private HashMap<Slot, JButton> buttons = new HashMap<Slot, JButton>();

    /** The colour to turn valid slots in the timetable when a module is selected. */
    private final Color HIGHLIGHT_COLOR = new Color(0xBBDEFB);

    /** The colour to turn slots which have a module scheduled in them. */
    private final Color SCHEDULED_COLOR = new Color(0x1976D2);

    /** The background colour of the timetable. */
    private final Color BACKGROUND_COLOR = new Color(0xA0BAD6);
```

```
33
    /** The color of the borders of the slots of the timetable. */
35  private final Color BORDER_COLOR = new Color(0x889db3);

37  //========================================================================
     // Constructor and helper methods
39   //========================================================================

41  /**
     * Instantiate the timetable view, by setting the layout, and adding buttons
43   * and labels.
     * @param slots the 2D array of {@link Slot}s in the timetable.
45   */
    public TimetableView(Slot[][] slots)
47  {
        // Initialize the panel with a grid layout.
49      // (extra row above and to the left for labels)
        super(new GridLayout(slots.length + 1, slots[0].length + 1));
51
        setBackground(BACKGROUND_COLOR);
53
        int rows = slots.length;
55      int cols = slots[0].length;

57      // When adding to a GridLayout, components fill along the row first.
        // Put blank JLabel in top left.
59      this.add(new JLabel());

61      // Add labels at the top for room names and sizes.
        for (int i = 0; i < cols; i++)
63          addTopLabel(slots[0][i].getName() + "\n" + slots[0][i].getSize());

65      // Fill out remaining rows with label on left, then buttons
        // To do this: loop through the rows
67      for (int i = 0; i < rows; i++)
        {
69          // Add JLabel to the left of each row.
            addLeftLabel(slots[i][0].getTime());
71
            // then fill out the rest of the row with buttons
73          for (int j = 0; j < cols; j++)
                addSlotButton(slots[i][j]);
75      }

77      // buttons disabled when program begins
        setButtonsEnabled(false);
79  }

81  /**
     * Adds a {@link JTextPane} label to the grid layout.
83   * This is used as it enabled multi-line text.
     * @param text the text to display in the label.
85   */
    private void addTopLabel(String text)
```

```
87      {
            // create text pane
89          JTextPane label = new JTextPane();
            label.setText(text);
91
            // set the text to be centered
93          StyledDocument doc = label.getStyledDocument();
            SimpleAttributeSet center = new SimpleAttributeSet();
95          StyleConstants.setAlignment(center, StyleConstants.ALIGN_CENTER);
            doc.setParagraphAttributes(0, doc.getLength(), center, false);
97
            // set colour and other properties of label, and add to layout
99          label.setForeground(Color.WHITE);
            label.setEditable(false);
101         label.setHighlighter(null);
            label.setBackground(BACKGROUND_COLOR);
103         label.setFont(new Font("Arial", Font.BOLD, 15));
            this.add(label);
105     }

107     /**
         * Adds a {@link JLabel} to the grid layout, with right aligned text.
109      * @param text the text to display on the label.
         */
111     private void addLeftLabel(String text)
        {
113         // create label with given text right aligned
            JLabel label = new JLabel(text, SwingConstants.RIGHT);
115
            // add invisible border so text has space
117         label.setBorder(new EmptyBorder(10, 10, 10, 10));

119         // set colour and font, and add to layout
            label.setForeground(Color.WHITE);
121         label.setFont(new Font("Arial", Font.BOLD, 15));
            this.add(label);
123     }

125     /**
         * Adds a button to the layout, storing a reference to its appropriate {@link Slot}
127      * alongside it.
         * @param slot the {@link Slot} for the button.
129      */
        private void addSlotButton(Slot slot)
131     {
            // create button
133         JButton button = new JButton();

135         // set style of button
            button.setBackground(Color.WHITE);
137         button.setForeground(Color.WHITE);
            button.setBorder(new LineBorder(BORDER_COLOR, 1));
139         button.setFont(new Font("Arial", Font.BOLD, 15));
            button.setFocusPainted(false);
```

```
141
            // add to buttons HashMap so button can be looked up via slot, then add to
143         // layout
            buttons.put(slot, button);
145         this.add(button);
        }
147
        //============================================================================
149      // Program methods
         //============================================================================
151
        /**
153      * Returns the {@link JButton} corresponding to a given slot in the timetable.
         * @param slot the {@link Slot} in the timetable.
155      * @return the {@link JButton}.
         */
157     public JButton getButtonAtSlot(Slot slot)
        {
159        return buttons.get(slot);
        }
161
        /**
163      * Set the text of the button in a given {@link Slot}.
         * @param slot the {@link Slot} to which the button corresponds.
165      * @param text the text which the button should display.
         */
167     public void setSlotText(Slot slot, String text)
        {
169        buttons.get(slot).setText(text);
        }
171
        /**
173      * Set the background colour of a slot button to the colour which
         * represents a scheduled module.
175      * @param slot the {@link Slot} to change.
         */
177     public void setSlotScheduledColor(Slot slot)
        {
179        buttons.get(slot).setBackground(SCHEDULED_COLOR);
        }
181
        /**
183      * Given an array of {@link Slot}s, turn their corresponding
         * buttons the highlight colour, which represents valid slots
185      * for a module to be placed into.
         * @param list the array of {@link Slot}s.
187      */
        public void highlightSlots(Slot[] list)
189     {
            // loop through the provided slots
191         for (int i = 0; i < list.length; i++)
            {
193            // look up button and highlight it
               buttons.get(list[i]).setBackground(HIGHLIGHT_COLOR);
```

```
195         }
        }
197
        /**
199      * Turn the background of all highlighted slot buttons back to white.
         */
201     public void clearHighlights()
        {
203        // loop through buttons and change background to white if it
           // is currently the highlight colour
205        JButton[] b = buttons.values().toArray(new JButton[0]);
           for (int i = 0; i < b.length; i++)
207           if (b[i].getBackground() == HIGHLIGHT_COLOR)
                 b[i].setBackground(Color.WHITE);
209
        }
211
        /**
213      * Toggle whether the slot buttons are enabled or disabled.
         * @param enabled whether the buttons should be enabled or disabled.
215      */
        public void setButtonsEnabled(boolean enabled)
217     {
           // loop through buttons and toggle enabled
219        JButton[] b = buttons.values().toArray(new JButton[0]);
           for (int i = 0; i < b.length; i++)
221           b[i].setEnabled(enabled);
        }
223 }
```

6 ModuleView.java

```
1  import java.awt.Color;
   import java.awt.Font;
3  import java.awt.GridLayout;
   import java.util.*;
5  import javax.swing.BorderFactory;
   import javax.swing.JButton;
7  import javax.swing.JPanel;

9  /**
    * Class to display the list of modules as part of the GUI.
11  * Modules are displayed as {@link JButtons}, with one
    * corresponding to each module. Includes methods to change
13  * colour and text of a given button.
    */
15 @SuppressWarnings("serial")
   public class ModuleView extends JPanel {
17
       //========================================================================
19      // Properties
        //========================================================================
21
        /**
```

```
23      * The buttons, as a {@link HashMap} enabling a button to be looked
        * up via its module.
25      */
        private HashMap<Module, JButton> buttons = new HashMap<Module, JButton>();
27
        /** The colour to turn the module button for the
29       * module which is being moved at the moment.
         */
31      private final Color HIGHLIGHT_COLOR = new Color(0xBBDEFB);

33      /** The colour to turn a module button once it has been
         * scheduled.
35       */
        private final Color SCHEDULED_COLOR = new Color(0x1976D2);
37
        /** The color of the borders of the slots of the timetable. */
39      private final Color BORDER_COLOR = new Color(0x889db3);

41      //============================================================================
        // Constructor and helper methods
43      //============================================================================

45      /**
         * Instantiate the module view from a given array of modules.
47       * @param modules the modules to display in the list.
         */
49      public ModuleView(Module[] modules)
        {
51         // set up grid layout
           super(new GridLayout(modules.length, 1));
53
           // add a button for each module to the grid layout
55         for (int i = 0; i < modules.length; i++)
              addButtonForModule(modules[i]);
57
           // buttons not enabled at the start of the program
59         setButtonsEnabled(false);
        }
61
        /**
63       * Add a module button to the view.
         * @param module the module for which to add a button.
65       */
        private void addButtonForModule(Module module)
67      {
           // create the button
69         JButton b = new JButton();

71         // set the style of the button
           b.setBackground(Color.WHITE);
73         b.setForeground(SCHEDULED_COLOR);
           b.setFont(new Font("Arial", Font.BOLD, 15));
75         b.setBorder(BorderFactory.createCompoundBorder(
                   BorderFactory.createLineBorder(BORDER_COLOR, 1),
```

```
77                BorderFactory.createEmptyBorder(10, 0, 10, 0)));
          b.setFocusPainted(false);
79
          // add button to view, and store in buttons HashMap
81        this.add(b);
          buttons.put(module, b);
83    }

85    //========================================================================================
       // Program Methods
87     //========================================================================================

89    /**
       * Get the button which corresponds to a given module.
91     * @param module the given module.
       * @return the {@link JButton} corresponding to it.
93     */
      public JButton getButtonForModule(Module module)
95    {
          // look up the button in the buttons HashMap
97        return buttons.get(module);
      }
99


101   /**
       * Change the colour of a module button to indicate that
103    * its module is being moved.
       * @param module the module to highlight.
105    */
      public void highlightModule(Module module)
107   {
          // get the button and change its style
109       JButton button = buttons.get(module);
          button.setBackground(HIGHLIGHT_COLOR);
111       button.setForeground(SCHEDULED_COLOR);
      }
113
      /**
115    * Change the colour of a module button to indicate that
       * its module has been scheduled.
117    * @param module the module which has been scheduled.
       */
119   public void makeScheduled(Module module)
      {
121       // get the button and change its style.
          JButton button = buttons.get(module);
123       button.setBackground(SCHEDULED_COLOR);
          button.setForeground(Color.WHITE);
125   }

127   /**
       * Change the colour of a module button to indicate that
129    * its module is no longer scheduled.
       * @param module the module which has been scheduled.
```

```
131      */
         public void makeUnscheduled(Module module)
133      {
            // get the button and change its style.
135         JButton button = buttons.get(module);
            button.setBackground(Color.WHITE);
137         button.setForeground(SCHEDULED_COLOR);
         }
139
         /**
141       * Update the text on a given module button.
          * @param module the module whose button needs changed.
143       * @param text the new text to display.
          */
145      public void setTextForButton(Module module, String text)
         {
147         buttons.get(module).setText(text);
         }
149
         /**
151       * Set all the buttons in the view to be either enabled
          * or disabled.
153       * @param enabled whether the buttons are to be enabled or not.
          */
155      public void setButtonsEnabled(boolean enabled)
         {
157         // get the buttons and set each en/dis-abled
            JButton[] b = buttons.values().toArray(new JButton[0]);
159         for (int i = 0; i < b.length; i++)
               b[i].setEnabled(enabled);
161      }
   }
```

7 ProgramView.java

```
   import javax.swing.*;
2  import java.awt.*;

4  /**
    * The view of the program. Responsible for laying out the GUI.
6   */
   @SuppressWarnings("serial")
8  public class ProgramView extends JFrame
   {
10
      //========================================================================
12     // Properties
       //========================================================================
14
      /** The timetable view. */
16    private TimetableView tv;

18    /** The module view. */
      private ModuleView mv;
```

```
20
    /** The background color of the GUI. */
22  private final Color BACKGROUND_COLOR = new Color(0xA0BAD6);

24  /** The dimensions of the screen. */
    private Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
26
    /** The edit button. */
28  private JButton editButton;

30  /**
     * Get the timetable view.
32   * @return the timetable view.
     */
34  public TimetableView getTimetableView()
    {
36     return tv;
    }
38
    /**
40   * Get the module view.
     * @return the module view.
42   */
    public ModuleView getModuleView()
44  {
       return mv;
46  }

48  /**
     * Get the edit button.
50   * @return the edit button.
     */
52  public JButton getEditButton()
    {
54     return editButton;
    }
56
    //========================================================================
58   // Constructor and setup methods
     //========================================================================
60
    /**
62   * Instantiate the GUI, by setting up the layout, adding components,
     * adding action listeners to buttons, and loading data from ModulesIn.txt.
64   */
    public ProgramView(Slot[][] slots, Module[] modules)
66  {
       setupBasicOptions();
68     setupGridBagLayout();
       layoutGUI(slots, modules);
70     this.setVisible(true);
    }
72
    /**
```

```
74      * Set the basic features of the GUI window, like its size and location.
        * Also sets up custom exit method for saving file on close.
76      */
    private void setupBasicOptions()
78  {
        getContentPane().setBackground(BACKGROUND_COLOR);
80      setTitle("Timetable Manager");
        setLocation(50,50);
82      // set do nothing on close so we can implement custom exit method
        setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
84      // set size based on screen size
        setSize((int) (
86          0.8 * screenSize.getWidth()),
            (int) (0.8 * screenSize.getHeight())
88      );
    }
90
    /**
92   * Set up the grid bag layout, which is used to arrange the GUI components.
     */
94  private void setupGridBagLayout()
    {
96      GridBagLayout gbl = new GridBagLayout();
        // there are four columns, the second and last are empty
98      // and just serve to separate module view from timetable and edge of gui
        gbl.columnWidths = new int[] {
100         (int) (getWidth() * 0.6),
            (int) (getWidth() * 0.05),
102         (int) (getWidth() * 0.2),
            (int) (getWidth() * 0.05)
104     };
        // sets how each column responds to being overfilled
106     gbl.columnWeights = new double[] { 1, 1, 1, 1 };
        // layout has 13 rows of equal height
108     gbl.rowHeights = new int[13];
        for (int i = 0; i < 13; i++)
110         gbl.rowHeights[i] = (int) (getHeight() / 13);
        // sets how each row responds to being overfilled
112     gbl.rowWeights = new double[] { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };
        // set the content pane to use this layout
114     getContentPane().setLayout(gbl);
    }
116
    /**
118  * Instantiate the program model and add GUI components.
     */
120 private void layoutGUI(Slot[][] slots, Module[] modules)
    {
122     addTimetableView(slots);
        addModulesLabel();
124     addScrollModuleView(modules);
        addEditButton();
126 }
```

```
128     /**
         * Adds the timetable view to the GUI.
130      * @param slots the 2D array of slots with which to create the timetable view.
         */
132     private void addTimetableView(Slot[][] slots)
        {
134        tv = new TimetableView(slots);
           addComponent(tv, 0, 1, 1, 11);
136     }

138     /**
         * Adds the modules label above the module view.
140      */
        private void addModulesLabel()
142     {
           // Create label and add to GUI.
144        JLabel label = new JLabel("Modules", SwingConstants.CENTER);
           label.setForeground(Color.WHITE);
146        label.setFont(new Font("Arial", Font.BOLD, 15));
           addComponent(label, 2, 1, 1, 1);
148     }

150     /**
         * Adds the module view, within a scroll pane, to the GUI.
152      * @param modules the array of modules with which to instantiate the module view.
         */
154     private void addScrollModuleView(Module[] modules)
        {
156        // create module view and scroll pane
           mv = new ModuleView(modules);
158        JScrollPane scroll = new JScrollPane();
           // set scroll pane parameters, and put module view inside it
160        scroll.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
           scroll.setViewportView(mv);
162        scroll.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
           scroll.setAlignmentX(LEFT_ALIGNMENT);
164        // this sets the scroll speed
           scroll.getVerticalScrollBar().setUnitIncrement(16);
166        // add to GUI
           addComponent(scroll, 2, 2, 1, 7);
168     }

170     /**
         * Add the edit/save button to the GUI.
172      */
        private void addEditButton()
174     {
           // add edit button
176        editButton = new JButton("START EDITING");
           // set style
178        editButton.setBackground(new Color(0xFFAF26));
           editButton.setForeground(Color.WHITE);
180        editButton.setFont(new Font("Arial", Font.BOLD, 15));
           editButton.setBorder(null);
```

```
182         editButton.setFocusPainted(false);
            // add to GUI
184         addComponent(editButton, 2, 10, 1, 1);
        }
186
        /**
188      * Add a component to the GUI.
         * @param comp the component to add.
190      * @param x the horizontal position within the grid bag layout.
         * @param y the vertical position within the grid bag layout.
192      * @param width the width in cells of the component.
         * @param height the height in cells of the component.
194      */
        private void addComponent(Component comp, int x, int y, int width, int height)
196     {
            // create grid bag constraints and add component
198         GridBagConstraints c = new GridBagConstraints();
            c.fill = GridBagConstraints.BOTH;
200         c.gridx = x;
            c.gridy = y;
202         c.gridwidth = width;
            c.gridheight = height;
204         getContentPane().add(comp, c);
        }
206
        //========================================================================
208      // Program methods
         //========================================================================
210
        /**
212      * Enable or disable the buttons in the view, and change the text of the
         * edit button.
214      * @param editEnabled true if the buttons should be enabled.
         */
216     public void toggleButtons(boolean enabled)
        {
218         mv.setButtonsEnabled(enabled);
            tv.setButtonsEnabled(enabled);
220         editButton.setText(enabled ? "SAVE CHANGES" : "START EDITING");
        }
222
        /**
224      * Set the text of a given module button.
         * @param module the module whose text needs set.
226      * @param text the text.
         */
228     public void loadUnscheduledModule(Module module, String text)
        {
230         mv.setTextForButton(module, text);
        }
232
        /**
234      * Called when a module is scheduled. Clears the text of module's previous slot,
         * updates the new slot, and clears highlighted slots. In addition, updates
```

```
236      * the module in the module view.
         * @param module the module that has been scheduled.
238      * @param slot the slot in which it has been placed.
         * @param text the description of the module.
240      * @param selectedSlot the module's previous slot, if any.
         */
242     public void scheduleModule(Module module, Slot slot, String text,
                                           Slot selectedSlot)
244     {
           if (selectedSlot != null)
246           tv.setSlotText(selectedSlot, "");
           // update text on slot button, clear the highlighted slots
248        // and make new slot scheduled color
           tv.setSlotText(slot, module.getCode());
250        tv.setSlotScheduledColor(slot);
           tv.clearHighlights();
252        // change module button background, and set text to module description
           mv.makeScheduled(module);
254        mv.setTextForButton(module, text);
        }
256
        /**
258      * Highlight a given module and its valid slots.
         * @param module the module.
260      * @param valid the slots.
         */
262     public void selectModule(Module module, Slot[] valid)
        {
264        // highlight the module and its valid slots, and store module.
           // note that text is not cleared from selected slot button, so
266        // that user is reminded which module they are moving, and where
           // it was previously scheduled
268        mv.highlightModule(module);
           tv.highlightSlots(valid);
270     }

272     /**
         * Set the style of a given module to be unscheduled, and set its text.
274      * Also clear highlighted slots.
         * @param module the module to be unscheduled.
276      * @param text the text description.
         */
278     public void makeUnscheduled(Module module, String text)
        {
280        mv.makeUnscheduled(module);
           mv.setTextForButton(module, text);
282        tv.clearHighlights();
        }
284 }
```

8 ProgramController.java

```
import javax.swing.*;
2 import java.awt.*;
```

```
import java.awt.event.*;

/**
 * The controller of the program. Responsible for
 * managing button presses and updating view (made up of timetable view and
 * module view) and model (program model) appropriately.
 */
public class ProgramController
{

   //================================================================================
    // Properties
    //================================================================================

   /** The model for the program, which stores all scheduling info. */
   private static ProgramModel model;

   /** The view for the program. */
   private ProgramView view;

   /** The module currently selected for scheduling. */
   private Module selectedModule;

   /** The slot in which the currently selected module was before it was selected. */
   private Slot selectedSlot;

   /** Whether editing the timetable is enabled. */
   private boolean editEnabled;

   //================================================================================
    // Constructor and setup methods
    //================================================================================

   /**
    * Instantiate the GUI, by setting up the layout, adding components,
    * adding action listeners to buttons, and loading data from ModulesIn.txt.
    */
   public ProgramController ()
   {
      model = new ProgramModel();
      view = new ProgramView(model.getSlots(), model.getModules());
      setupTimetableButtons();
      setupModuleButtons();
      setupEditButton();
      setupQuitButton();
      loadData();
   }

   /**
    * Adds an {@link ActionListener} to each slot button in the timetable
    * view, so that slotPressed is called when the button is pressed.
    */
   private void setupTimetableButtons()
   {
```

```
        Slot[][] slots = model.getSlots();

        // loop over slots and add action listener to matching button
        for (int i = 0; i < slots.length; i++)
            for (int j = 0; j < slots[0].length; j++)
            {
                final Slot slot = slots[i][j];
                view.getTimetableView().getButtonAtSlot(slot).addActionListener(
                    new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                            slotPressed(slot);
                        }
                    }
                );
            }
    }

    /**
     * Adds an {@link ActionListener} to each module button in the module
     * view, so that modulePressed is called when the button is pressed.
     */
    private void setupModuleButtons()
    {
        Module[] modules = model.getModules();

        // loop over slots and add action listener to matching button
        for (int i = 0; i < modules.length; i++)
        {
            final Module module = modules[i];
            view.getModuleView().getButtonForModule(module).addActionListener(
                    new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                            modulePressed(module);
                        }
                    });
        }
    }

    /**
     * Adds an {@link ActionListener} to the edit/save button, so that
     * editPressed is called when it is pressed.
     */
    private void setupEditButton()
    {
        view.getEditButton().addActionListener(
            new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    editPressed();
                }
            });
    }

    /**
     * Set the program to call the save to file method
```

```
     * when the gui is closed, then quit the program.
     */
    private void setupQuitButton()
    {
        // add custom window close method
        view.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent event){
                saveToFile();
                System.exit(0);
            }
        });
    }

    /**
     * Load the schedule data from the timetable into the views
     * (both timetable and module views).
     */
    private void loadData()
    {
        // get the slots which have a module in them
        Slot[] filledSlots = model.getFilledSlots();
        // load each slot into view (text field blank as text is set below)
        for (int i = 0; i < filledSlots.length; i++)
            view.scheduleModule(
                  model.moduleInSlot(filledSlots[i]),
                  filledSlots[i],
                  "",
                  null
              );

        // get all the modules
        Module[] ms = model.getModules();
        // for each module
        for (int i = 0; i < ms.length; i++)
            view.loadUnscheduledModule(ms[i], model.lineForModule(ms[i]));
    }

    //========================================================================
     // Button press methods
     //========================================================================

    /**
     * Called when the edit/save button is pressed. Enabled or disable view buttons
     * accordingly, and change edit button text. In addition, save changes if
     * appropriate.
     */
    public void editPressed()
    {
        // enable or disable buttons, and update edit button
        view.toggleButtons(!editEnabled);

        // save if necessary
        if (editEnabled)
        {
```

```
            // important to clear the selected module before buttons are disabled
166         deselectModule();
            saveToFile();
168     }

170     // toggle edit enabled
        editEnabled = editEnabled ? false : true;
172 }

174 /**
     * Called whenever a slot button is pressed. Updates model and view
176  * appropriately.
     * @param slot the slot that was pressed.
178  */
    public void slotPressed(Slot slot)
180 {
        // schedule selected module if one is selected
182     if (selectedModule != null)
            // if the selected module is successfully scheduled, return
184         if (scheduleModule(selectedModule, slot))
                return;
186     // if no module selected, or module wasn't successfully scheduled,
        // and if there is a module in the slot
188     if (model.moduleInSlot(slot) != null)
            // act as if the module button for that module was pressed
190         modulePressed(model.moduleInSlot(slot));
        // otherwise, do nothing
192 }

194 /**
     * Called whenever a module button is pressed. Updates model and view
196  * appropriately.
     * @param module
198  */
    public void modulePressed(Module module)
200 {
        // if there is a module selected
202     if (selectedModule != null)
            // if pressed module is the selected module, de-select it
204         if (selectedModule == module)
                deselectModule();
206         else
            {
208             // pressing a different module de-selects selected module
                // and selects new module instead
210             deselectModule();
                selectModule(module);
212         }
        // if no module is currently selected, select the pressed module
214     else
            selectModule(module);
216 }

218 //============================================================================
```

```
    // Scheduling helper methods
    //========================================================================

    /**
     * Adds a module to a given slot in the timetable, provided the slot is a valid
     * place to put the module, and updates views to reflect the change.
     * @param module the module to schedule.
     * @param slot the slot in which to put it.
     * @return whether the module was scheduled.
     */
    private boolean scheduleModule(Module module, Slot slot)
    {
        // check if slot is valid place to put module
        if (model.moduleFitsInSlot(module, slot))
        {
            // update model
            model.addModuleToSlot(module, slot);
            // and view
            view.scheduleModule(
                    module,
                    slot,
                    model.lineForModule(module),
                    selectedSlot
                );
            // clear selected module and return true since module
            // was successfully scheduled
            selectedModule = null;
            return true;
        }
        // if not valid slot, return false
        return false;
    }

    /**
     * Select a given module, and highlight the slots in the
     * timetable view into which the module may be placed.
     * @param module the module to be selected.
     */
    private void selectModule(Module module)
    {
        // save the slot in which the module is currently scheduled
        selectedSlot = model.slotForModule(module);
        // clear that slot in timetable if not null
        if (selectedSlot != null)
            model.addModuleToSlot(null, selectedSlot);
        // update the view
        view.selectModule(module, model.validSlotsForModule(module));
        selectedModule = module;
    }

    /**
     * Deselect the currently selected module - put it back into its previous slot if
     * there was one, otherwise keep it unscheduled, and update views accordingly.
     */
```

```
    private void deselectModule()
    {
        // need to save reference to selected module, as calling
        // scheduleModule makes selected module null
        Module module = selectedModule;
        // if no module is selected, there is nothing to be done
        if (module == null)
            return;
        // if the selected module was previously scheduled
        if (selectedSlot != null)
            // reschedule it in its previous slot
            scheduleModule(module, selectedSlot);
        else
            // otherwise, return the selected module button to its unscheduled style
            view.makeUnscheduled(module, model.lineForModule(module));

        selectedModule = null;
    }

    //========================================================================================
     // File saving
     //========================================================================================

    /**
     * Save the changes to the output file, and show a message to the user.
     */
    private void saveToFile()
    {
        // only save if necessary
        if (!editEnabled)
            return;
        model.saveToFile();
        // create and display message to user
        UIManager.put("OptionPane.background", Color.WHITE);
        UIManager.put("Panel.background", Color.WHITE);
        JOptionPane.showMessageDialog(
              null,
              "Changes saved to ModulesOut.txt.",
              "Changes Saved",
              JOptionPane.INFORMATION_MESSAGE
        );
    }
}
```