Criterion C: Development

The product was developed according to the design and was improved based on the existing program. The following complexities were used in the project:

1. hierarchical composite data structure
2. Inheritance
3. Timer synchronization
4. Searching algorithm
5. Persistence library – xstream
6. GUI design
7. Bilingual implementation

The program is divided into 4 fundamental packages: frame, model, persistence, utilities. Frame is for interface switching and GUI designs. In “Frame”, each interface is created as a panel, inherited from the *AbstractPanel* class. Classes in the *Model* package manages the data stracture. *Persistance* package is for data read and written. At last, *utilities* package includes the classes for creating the Timer and the Warning Tone in the program. Seperating this utility function helps to improve the reusebility and readbility of the code.

UML Class Diagram (Appendix-UML diagram for a clear image):

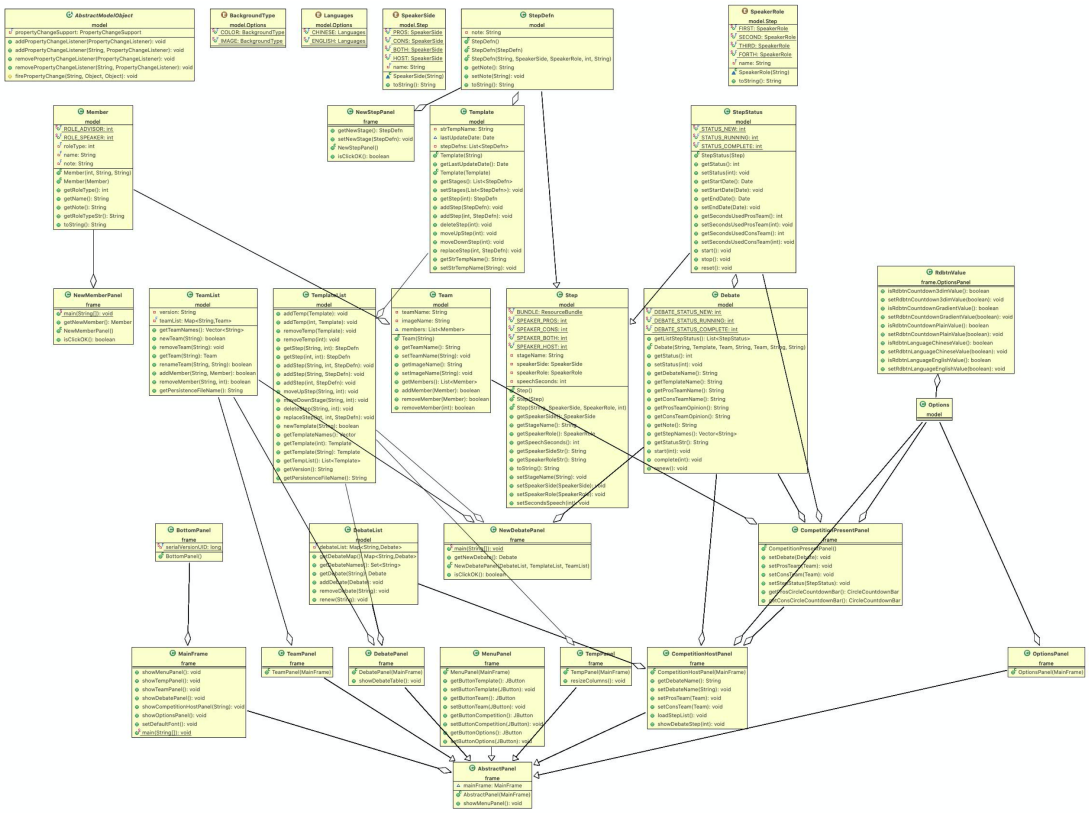


Figure 0.1_UML Class Diagram

1. hierarchical composite data structure – linkedlist in linkedHashmap

Hierarchical composite data structure is used repeatedly in the program to manage the data. Three types of ADTs are used in the program: ArrayList, LinkedHashmap, Hashmap. For instance, *Step*, *Template, TemplateList* form a hierarchical composite data structure.

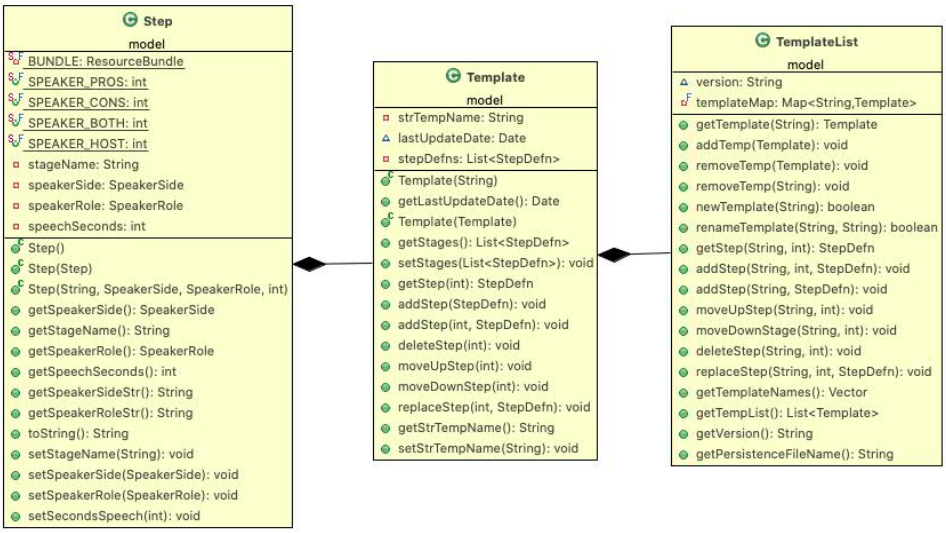


Figure 1.1_UML TemplateList, Template, Step hierarchical composite data structure

```
11 public class Template{
12     private String strTempName;
13     Date lastUpdateDate = new Date();
14     private List<StepDefn> stepDefns = new ArrayList<StepDefn>();
15
16     public Template(String strTempName) throws Exception{
17         if (strTempName == null || strTempName.equals("")) {
18             throw new Exception("Template name is null");
19         }
20         this.strTempName = strTempName;
21     }
22
23     public Date getLastUpdateDate() {
24         return lastUpdateDate;
25     }
26
27     public Template(Template template) {
28         this.strTempName = template.getStrTempName();
29         for(Iterator<StepDefn> iterator=template.stepDefns.iterator(); iterator.hasNext(); ) {
30             StepDefn stepDefn = (StepDefn)iterator.next();
31             StepDefn newStep = new StepDefn(stepDefn);
32             this.stepDefns.add(newStep);
33         }
34     }
```

Template class: creating a ArrayList that stores `StepDefn` objects

Figure 1.2_code_Template class constructor

Reasons for using ArrayList for **Template**:

a. ArrayList is a dynamic data structure, automatic length setting of the array.
b. Sequence access to the items (`step`) in the ArrayList, satisfying the need to go through the debate `steps` one by one.
c. ArrayList allows objects with the same name. e.g. there could be more than one `step` called “proposing”.

```
21 public class TemplateList implements IPersistable{
22     String version = "version 1.0";
23
24     private final Map<String, Template> templateMap = new LinkedHashMap<String, Template>();
25
26     public Template getTemplate(String templateName) {
27         return templateMap.get(templateName);
```

refer to the *persistence* package

access to the item within the LinkedHashMap directly with the **key** (templatName)

Figure 1.3_code_TemplateList class

LinkedHashMap contains the characteristics of both LinkedList and HashMap. Using LinkedHashMap to store `Templates` brings several benefits:

a. Hashmap data structure has "key" to identify each item within the Hashmap. The "key" here is the `templateName` (as shown above in the blue box) which acts as a signature of each item in the map. Then same key signature cannot appear twice, which avoids the situation that the same item is inserted twice; LinkedHashMap allows directly access to the item within the data structure.

A Graph showing the HashMap data structure:

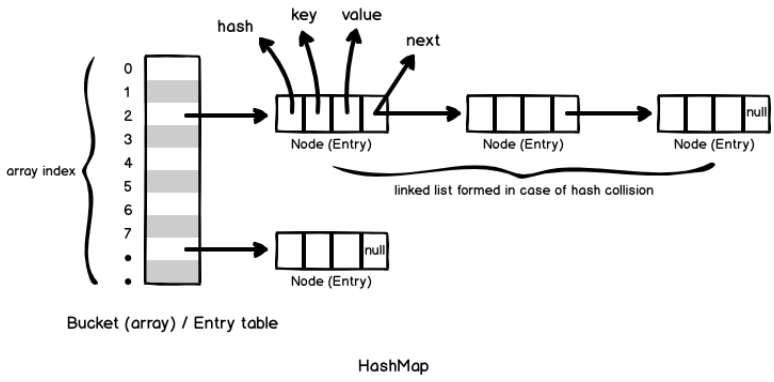
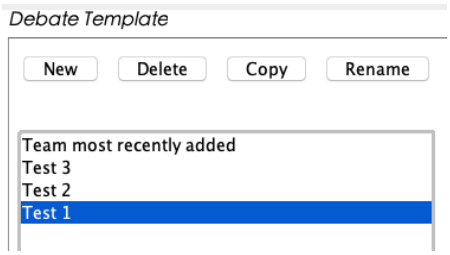


Figure 1.4_abstract HashMap diagram_*citation[1]



`Templates` are sorted according to the time it created; a stack structure.

Figure 1.5_interface_Template List

b. LinkedList property allows sorting within the list. In this program, this property LinkedList is utilized to sort the `template` in the `templateList` (as shown in the

[1] "How HashMap Works Internally in Java?" Java by Examples, www.javaquery.com/2019/11/how-hashmap-works-internally-in-java.html.

figure above) according to the chronological order; the most recently added **template** will show on the top.

A Graph showing the LinkedList data structure:

Figure 1.6_abstract LinkedList diagram

```
111  public Vector getTemplateNames() {
112      Vector<String> vector = new Vector<String>();
113
114      // add template names to vector
115      for(String name:templateMap.keySet()) {
116          vector.add(0, name);
117      }
118
119      return vector;
120  }
```

Add the new **Template** at the beginning of the LinkedHashMap

Figure 1.7_code_ adding Template in the TemplateList

A Graph showing the data structure of *LinkedHashMap:*

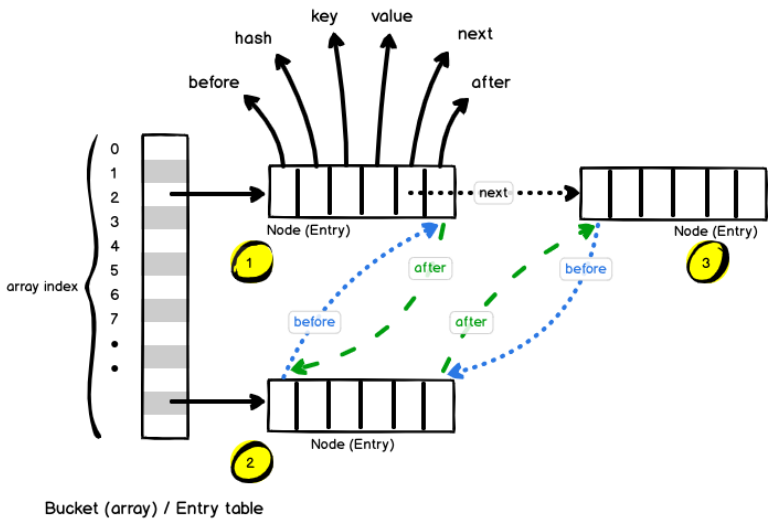


Figure 1.8_abstract LinkedHashMap diagram_*citation[2]

[2] "How LinkedHashMap Works Internally in Java?" Java by Examples, www.javaquery.com/2019/12/how-linkedhashmap-works-internally-in.html.

A Graph showing the data structure of *Member-Team-TeamList* within the IA program *(Team: Arraylist, TeamList: LinkedHashMap):*

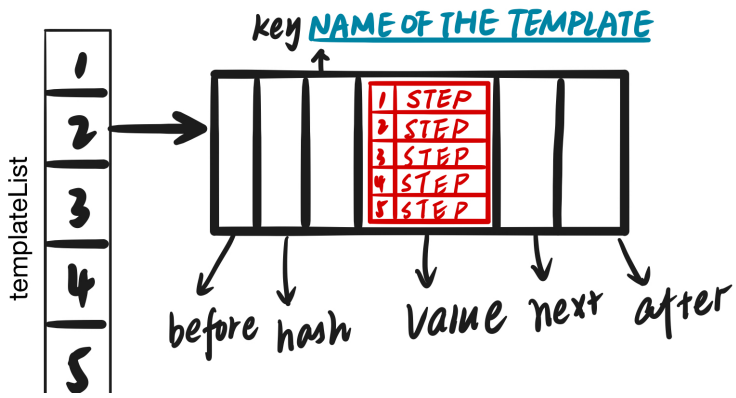


Figure 1.9_ hierarchical composite data structure-ArrayList within LinkedHashmap

Similar logic has applied in hierarchical data structure for `Member-Team-teamList`, which forms a List(`Team`) with in a LinkedHashMap(`teamList`):

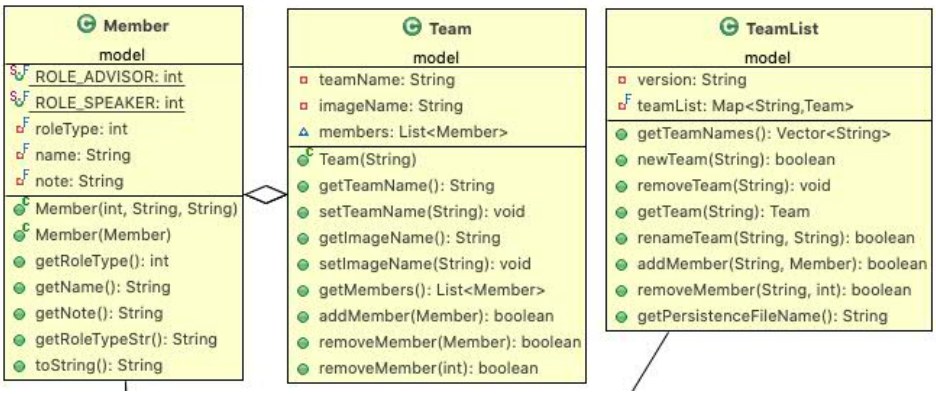


Figure 1.10_UML TeamList, Team, Member hierarchical composite data structure

```
 9 public class Team {
10
11     private String teamName;
12     private String imageName;
13     List<Member> members;
```

Team class, uses List to store **members**, allowing same name appears twice in the List.

```
15 public class TeamList implements IPersistable{
16     private String version = "version 1.0";
17
18     private final Map<String, Team> teamList
19     = new LinkedHashMap<String, Team>();
```

TeamList class, uses LinkedHashMap to store **team**. Avoiding two teams with the same name; allowing team sorting.

Figure 1.10_UML TeamList, Team, Member hierarchical composite data structure

`debateList` and `Debate` also form a hierarchical composite data structure. Similarly, a `debateList` is a collection of `Debate`. A `Debate` object is composed of Team and Template. Overall, hierarchical composite data structure can be seen everywhere in the program, which provide a more flexible data management.

2. Inheritance

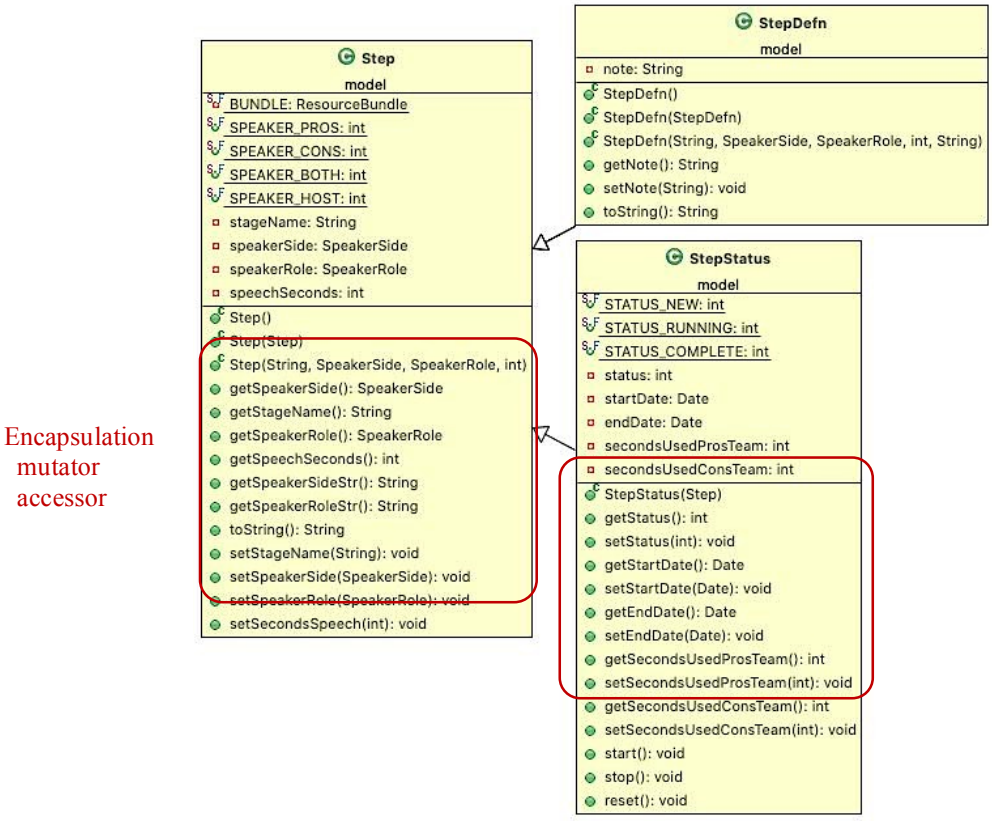


Figure 2.1_Step_Inheritance_UML diagram

Inheritance is used twice in my program. The UML above shows the relationship between parent class `Step` and its child class `StepStatus` and `StepDefn`.

```
public class Step{
    private static final ResourceBundle BUNDLE = ResourceBundle.getBundle("frame.messages");

    public static final int SPEAKER_PROS=1;
    public static final int SPEAKER_CONS=2;
    public static final int SPEAKER_BOTH=3;
    public static final int SPEAKER_HOST=4;

    private String stageName;
    private SpeakerSide speakerSide;
    private SpeakerRole speakerRole;
    private int speechSeconds;

    static public enum SpeakerSide {
        PROS("Pros Team"),
        CONS("Cons Team"),
        BOTH("Both Teams"),
        HOST("Host");

        private final String name;

        SpeakerSide(String string){
            this.name = string;
        }

        @Override
        public String toString() {
            return name;
        }
    }
```

enum: The value of SpeakerSide is limited 4

Figure 2.2_code_parent class Step_1

`Step` class

```
static public enum SpeakerRole {
    FIRST("First Speaker"),
    SECOND("Second Speaker"),
    THIRD("Third Speaker"),
    FORTH("Forth Speaker");

    private final String name;

    SpeakerRole(String string){
        this.name = string;
    }

    @Override
    public String toString() {
        return name;
    }
}

public Step(Step stepDefn) {
    super();
    this.stageName = stepDefn.stageName;
    this.speakerSide = stepDefn.speakerSide;
    this.speakerRole = stepDefn.speakerRole;
    this.speechSeconds = stepDefn.speechSeconds;
}

public Step(String stageName, SpeakerSide speakerSide, SpeakerRole speakerRole, int speechSeconds) {
    this.speakerSide = speakerSide;
    this.stageName = stageName;
    this.speakerRole = speakerRole;
    this.speechSeconds = speechSeconds;
}
```

enum: The value of SpeakerRole is limited 4.

The *Step* class includes 4 main variables; override constructor

Figure 2.3_code_parent class Step_2

`StepStatus` class

```
public class StepStatus extends Step{

    public static final int STATUS_NEW=0;
    public static final int STATUS_RUNNING = 1;
    public static final int STATUS_COMPLETE = 2;

    private int status=STATUS_NEW;
    private Date startDate=null;
    private Date endDate=null;
    private int secondsUsedProsTeam=0;
    private int secondsUsedConsTeam=0;

    public StepStatus(Step step) {
        super(step);
    }
```

Adding a new attribute, step status, `status`

Figure 2.4_code_StepStatus class

***StepDefn* class**

```
public class StepDefn extends Step{
    private String note;

    public StepDefn() {
        super();
    }

    public StepDefn(StepDefn stepDefn) {
        super(stepDefn);
        this.note = stepDefn.getNote();
    }

    public StepDefn(String stageName, SpeakerSide speakerSide,
            SpeakerRole speakerRole, int speechSeconds, String note) {
        super(stageName, speakerSide, speakerRole, speechSeconds);
        this.note = note;
    }
```

Adding a new attribute, step note. It is used when creating a new step

Figure 2.5_code_StepDefn class

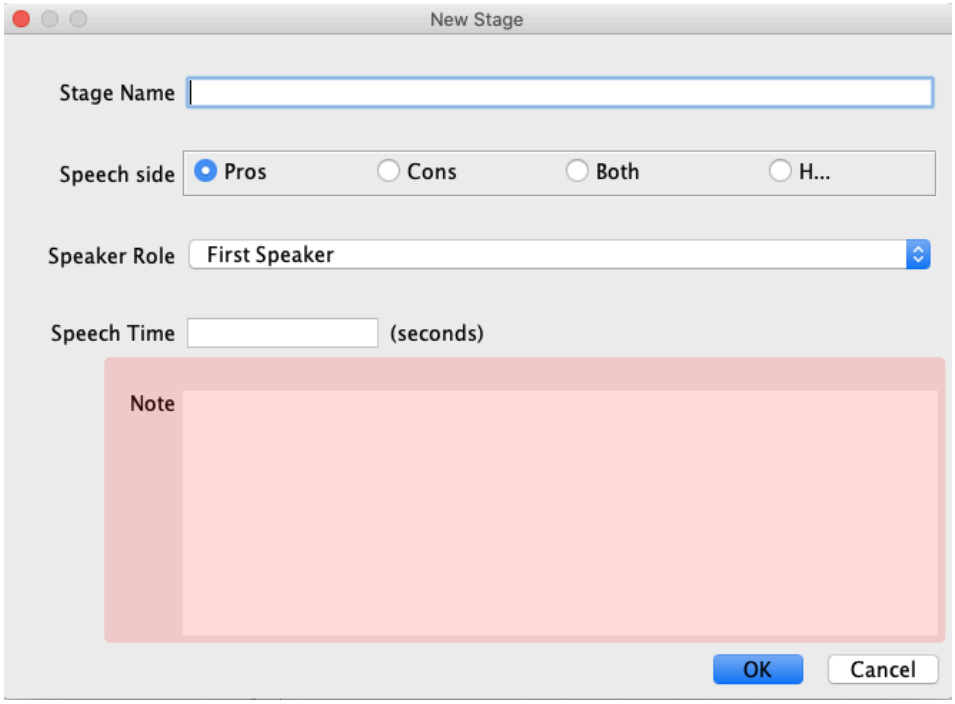


Figure 2.4_demonstration_StepDefn class_ note attribute

3. Timer synchronization (ActionListener)

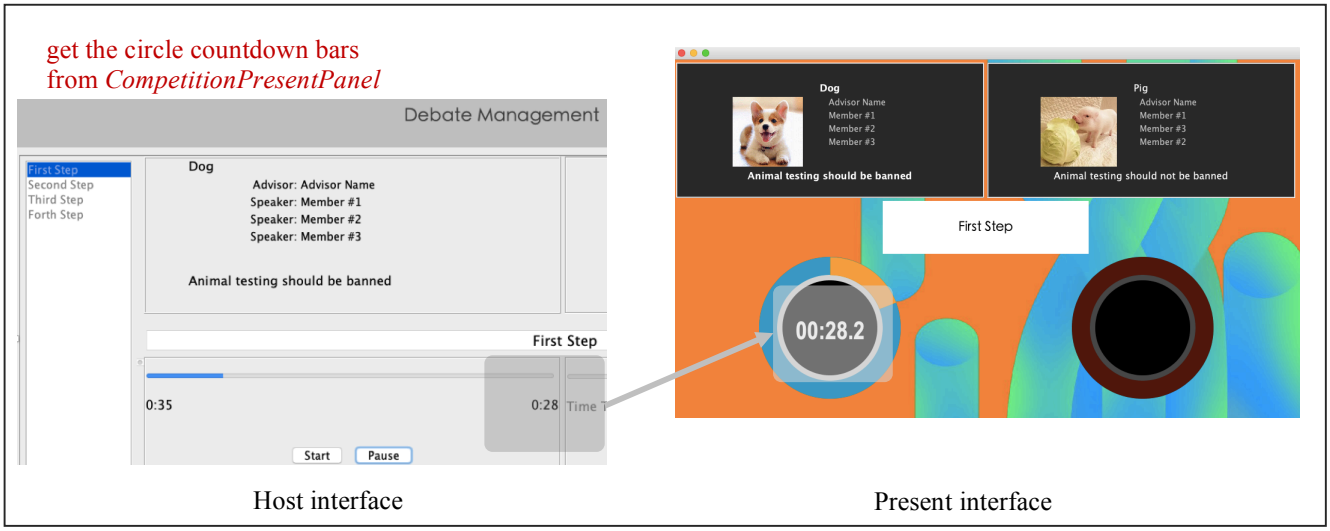


Figure 3.1_Timer synchronization on double screens

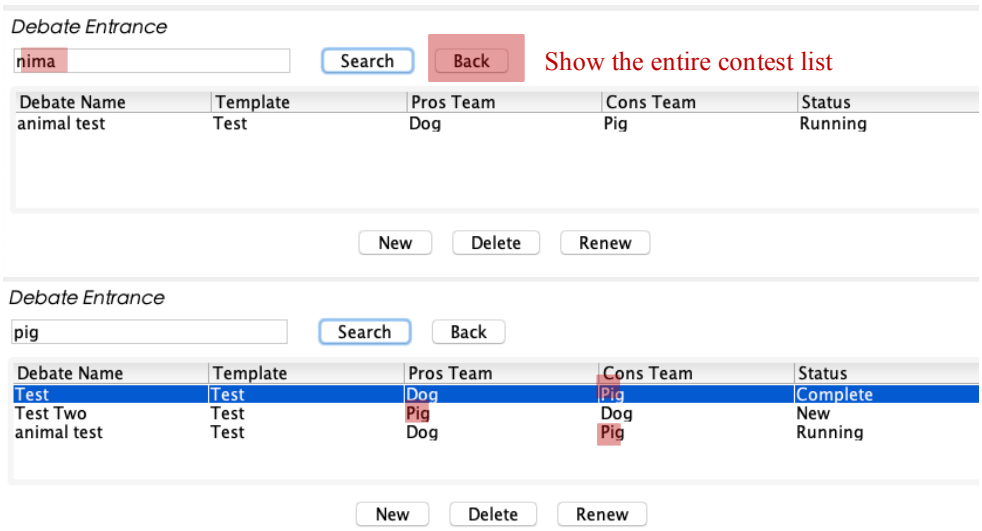
Figure 3.2_Timer_ActionListener

The ActionListener of the `consCircleCountdownBar` has the same logic as the that of `prosCircleCountdownBar`. It used the Observer logic in the development, getting the dynamic variable from one class to another.

4. Searching algorithm

In the Debate Entrance page, users could search specific debate contest(s) by entering the key word. The searching function here makes use of the fundamental linear searching algorithm and several string methods (concatenation, toLowerCase, SubString).

Even if the user does not type in the entire name of the debate



It shows all the debates that include key word "pig"

Figure 4.1 Searching_Interface display

```
468 btnSearch.addActionListener(new ActionListener() {
469     public void actionPerformed(ActionEvent e) {
470         showDebateTable();
471         updateUI();
472     }
473 });
474
475 btnBack.addActionListener(new ActionListener() {
476     public void actionPerformed(ActionEvent e) {
477         txtSearch.setText("");
478         showDebateTable();
479         updateUI();
480     }
481 });
```

ActionListener to **Search** and **Back** button: call showDebateTable() method

Empty the text field

Figure 4.2_Searching button ActionListener_code

In the *DebatePanel* class, showDebateTable() method is called if there is any modifications to the contest list.
showDebateTable() Method:

```
617 public void showDebateTable() {
618     Vector<String> columnNames = new Vector<String>();
619     columnNames.add("Debate Name");
620     columnNames.add("Template");
621     columnNames.add("Pros Team");
622     columnNames.add("Cons Team");
623     columnNames.add("Status");
624     Vector<Vector<String>> rowData = new Vector<Vector<String>>();
```

Assigning column names to the table

Figure 4.2_Searching algorithm code 1

```
626 if (debateList != null) {
627     for (Entry<String, Debate> entry : debateList.getDebateMap().entrySet()) {
628         String key = entry.getKey();
629         Debate debate = entry.getValue();
```

Loop over the debate map

Get **debate** objects from the map

Figure 4.3_Searching algorithm code 2

```
631 if (txtSearch.getText() != null && !txtSearch.getText().equals("")) {
632     String search = txtSearch.getText().toLowerCase();
633     int len = search.length();
634     String concate = "";
635     concate = debate.getDebateName() + "∫" + debate.getTemplateName() + "∫"
636             + debate.getProsTeamName()
637             + "∫" + debate.getConsTeamName() + "∫" + debate.getStatusStr();
638     concate = concate.toLowerCase();
639     for (int i = 0; i < concate.length() - len - 1; i++) {
640         int j = i + len;
641         String frac = concate.substring(i, j);
642         if (frac.equals( search)) {
643             Vector<String> row = new Vector<String>();
644             row.add(debate.getDebateName());
645             row.add(debate.getTemplateName());
646             row.add(debate.getProsTeamName());
647             row.add(debate.getConsTeamName());
648             row.add(debate.getStatusStr());
649             rowData.add(row);
650             break;
```

if statement: Determine whether there is text in the text area

Get the length of the user input `len`

String concatenation

Get the substring (of length `len`) of the concatenated information, compare it with the user input

Display the matched `contest` objects on the table

Figure 4.4_Searching algorithm code 3

User input:
Nimal
nimal (to lower case)

The length of the input **len**: nimal.length() = 5

Concatenated information of a contest:
Animal test∫Test∫Dog∫Pig
animal test∫test∫dog∫pig (to lower case)

Substring linear search. Substring (i,j). j-i = **len**

Substring (0,5)
#1: animal test∫test∫dog∫pig (false)

Substring (1,6)
#2: animal test∫test∫dog∫pig (true)

Using "∫" to connect the strings is to avoid the situation that adjacent variables could form a word.

The Flow Chart of the Searching Algorithm can be found in **Criterion B_Design**, *Flowchart & Pseudocode*

Figure 4.5_Searching algorithm Demonstration

In the code of the **Back button** above, after clicking **Back**, the actionlistener will empty the `txtSearch`. Therefore, the outmost if statement is false, instead of showing the conditional contest list, the table will show the full list. toLowerCase() is used twice in the code above, converting both the user input and the information in the table to lowercase for comparison.

```
648 } else {
649     //display the full debate item that does not contain the key word/key word is empty
650     Vector<String> row = new Vector<String>();
651     row.add(debate.getDebateName());
652     row.add(debate.getTemplateName());
653     row.add(debate.getProsTeamName());
654     row.add(debate.getConsTeamName());
655     row.add(debate.getStatusStr());
656     rowData.add(row);
657 }
```

If the search text field is empty, the algorithm will loop over the `debate` contest map and show all the existing Debate contest on the table

Figure 4.6_Searching algorithm code 4

5. Persistence library – xstream

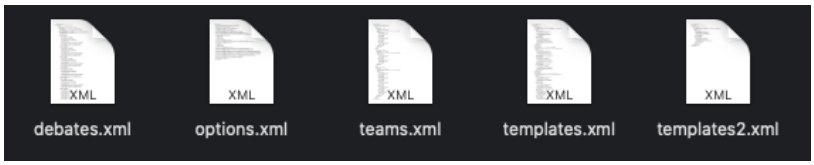


Figure 5.1_local xml files

XStream is a simple library to serialize objects to XML and back again. The program makes use of XStream to save the object as local xml files automatically.

```
1 package persistence;
2
3 public interface IPersistable {
4     public String getPersistenceFileName();
5 }
```

Figure 5.2_code_persistence

General	Detail
Performance	Fast, save storage space
Easy to read	Clear and organized xml files
Object supported	Serialized the internal field of the object

Table 5.3_Benefits of introducing XStream

6. GUI design

window-builder plug-in and several awt and swing libraries are used in the program.

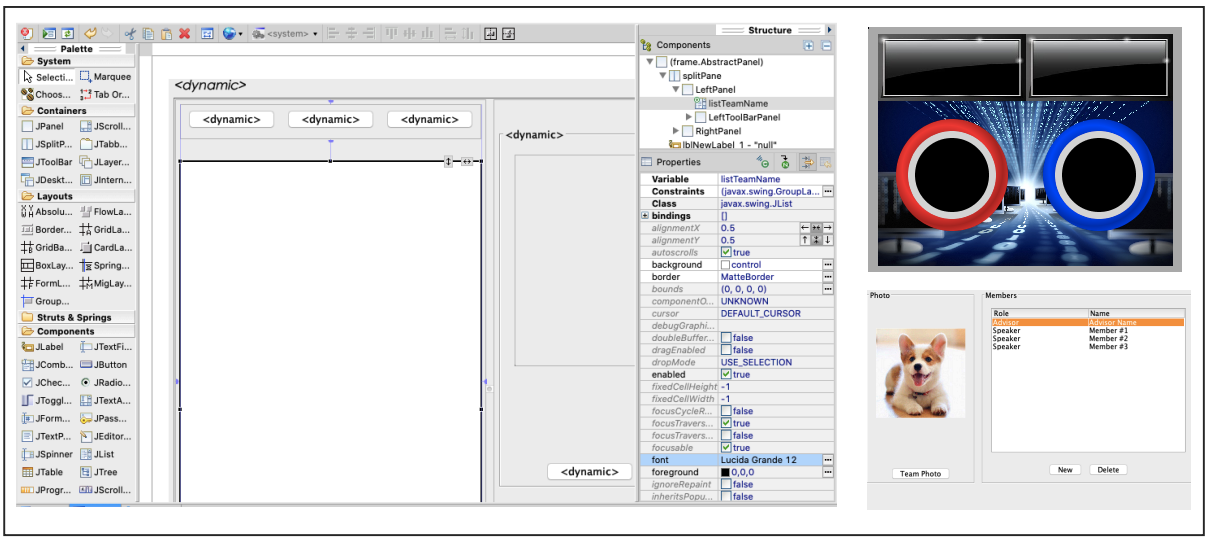


Figure 6.2_GUI design

7. Bilingual implementation

When assigning the text for labels and buttons in the program, I referred to the properties files as shown below.

messages_en_US.properties

```
All.Cancel=\u53D6\u6D88
All.OK=\u786E\u8BA4
All.Return=\u8FD4\u56DE
MenuPanel.buttonCompetition.text=\u8D5B\u4E8B\u7BA1\u7406
MenuPanel.buttonOptions.text=\u7CFB\u7EDF\u8BBE\u7F6E
MenuPanel.buttonTeam.text=\u53C2\u8D5B\u961F\u4F0D\u767B\u8BB0
MenuPanel.buttonTemplate.text=\u8D5B\u7A0B\u6A21\u677F\u5B9A\u4E49
NewStepPanel.cancelButton.text=\u53D6\u6D88
NewStepPanel.lblNewLabel.text=\u53D1\u8A00\u65B9
NewStepPanel.lblNewLabel_1.text=\u53D1\u8A00\u65F6\u95F4
NewStepPanel.lblNewLabel_2.text=\u5907\u6CE8
NewStepPanel.lblSeconds.text=\uFF08\u79D2\uFF09
NewStepPanel.lblSpeakerName.text=\u53D1\u8A00\u4EBA
NewStepPanel.lblSpeakerRole.text=\u53D1\u8A00\u89D2\u8272
NewStepPanel.lblStageName.text=\u8D5B\u6BB5\u540D\u79F0
NewStepPanel.okButton.text=\u786E\u8BA4
NewStepPanel.rdbtnBoth.text=\u53CC\u65B9
NewStepPanel.rdbtnCons.text=\u53CD\u65B9
NewStepPanel.rdbtnHost.text=\u4E3B\u6301\u4EBA
NewStepPanel.rdbtnPros.text=\u6B63\u65B9
TeamPanel.btnDelete.text=\u5220\u9664
TeamPanel.btnDeleteTemplate.text=\u5220\u9664
TeamPanel.btnNew.text=\u65B0\u5EFA
TeamPanel.btnNewButton.text=\u8FD4\u56DE
TeamPanel.btnNewTemplate.text=\u65B0\u5EFA
TeamPanel.btnRename.text=\u8BE5\u540D
TeamPanel.btnSelectPhoto.text=\u961F\u4F0D\u7167\u7247
TeamPanel.lblNewLabel_1.text=\u53C2\u8D5B\u961F\u5B9A\u4E49
```

messages_zh_CN.properties

```
All.Cancel=Cancel
All.OK=OK
All.Return=Return
MenuPanel.buttonCompetition.text=Debate Entrance
MenuPanel.buttonOptions.text=System Options
MenuPanel.buttonTeam.text=Debate Team
MenuPanel.buttonTemplate.text=Debate Template
NewStagePanel.cancelButton.text=Cancel
NewStagePanel.lblNewLabel.text=Speech side
NewStagePanel.lblNewLabel_1.text=Speech Time
NewStagePanel.lblNewLabel_2.text=Note
NewStagePanel.lblSeconds.text=(seconds)
NewStagePanel.lblSpeakerName.text=Speaker Name
NewStepPanel.lblSpeakerRole.text=Speaker Role
NewStagePanel.lblStageName.text=Stage Name
NewStagePanel.okButton.text=OK
NewStagePanel.rdbtnBoth.text=Both
NewStagePanel.rdbtnCons.text=Cons
NewStagePanel.rdbtnHost.text=Host
NewStagePanel.rdbtnPros.text=Pros
TeamPanel.btnDelete.text=Delete
TeamPanel.btnDeleteTemplate.text=Delete
TeamPanel.btnNew.text=New
TeamPanel.btnNewButton.text=Return
TeamPanel.btnNewTemplate.text=New
TeamPanel.btnRename.text=Rename
TeamPanel.btnSelectPhoto.text=Team Photo
TeamPanel.lblNewLabel_1.text=Debate Team
```

Figure 6.1_language messages files

```
 9 public class Options {
10     public enum Languages{CHINESE, ENGLISH};
```

Enumeration type variables in *Option* class, including CHINESE and ENGLISH

Figure 6.2_code_ language option

For example, in the *MenuPanel* class, instead of giving a final String text value to the **Jbutton**, the text is acquired from the **messages files**:

```
32 JButton buttonTemplate = new JButton(Messages.getString("MenuPanel.buttonTemplate.text"));
33 JButton buttonTeam = new JButton(Messages.getString("MenuPanel.buttonTeam.text"));
34 JButton buttonCompetition = new JButton(Messages.getString("MenuPanel.buttonCompetition.text"))
35 JButton buttonOptions = new JButton(Messages.getString("MenuPanel.buttonOptions.text"));
```

Figure 6.3_code_ Jbutton text setting_messages files

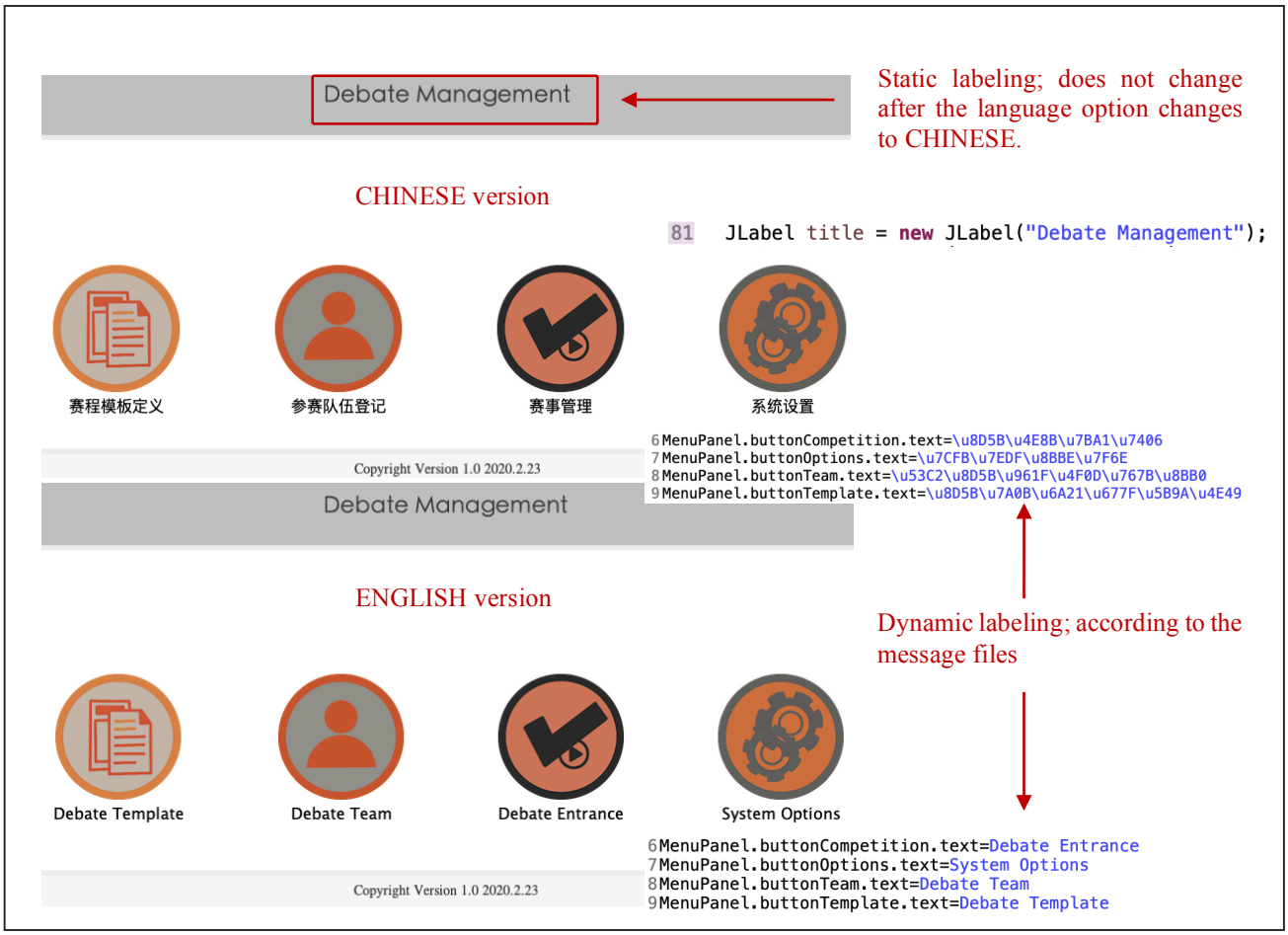


Figure 6.4_demonstration_ Menu_ Jbutton double languages

Referenced External Library:

jgoodies, xstream, beansbinding

Library & Source Citation:

[1] "Professional Java Desktop." JGoodies, www.jgoodies.com/.

[2] Gson 2.8.6 Javadoc (Com.google.code.gson), www.javadoc.io/doc/com.google.code.gson/gson.

[3] "About XStream." XStream - About XStream, x-stream.github.io/.

[4] NetBeans, Apache. "Apache NetBeans 11.3." Binding Beans and Data in a Java Application, netbeans.apache.org/kb/docs/java/gui-binding.html.