```java
package project;

import java.awt.Color;

/* This class contains necessary data to construct a Form base Interface.
 * The data describe the Input Fields(JTextField,JRadioButton,JComboBox..etc), Labels for the Fields,
 * Buttons, Prompt Messages & Error Messages to the User, Input Fields Validators, Input Fields sizes,
 * Directions of the Radio Button group or Check boxes (possible directions are Vertical or Horizontal),
 * Locations of the Buttons position (Whether at the bottom of the form or Side),
 * Options values(display and actual values) for RadioButtons or ComboBoxes etc, Input Filed's Class Type,
 * Color of the Form, Actual data of the entity that it represent (Data of the Inputs) */
public class FormConfiguration implements FormSaveListener {

    private String[] labelNames;        //Contains all the Labels of the Underlying Form Interface
    private String[] fieldNames;        //Contains all the Input Fields of the Underlying Form Interface
    private Class<?>[] fieldClass;      //Contains all Class Types(Input Fields) of the Underlying Form Interface
    private int[] fieldSize;            //Contains all the Input Filed Size of the Underlying Form Interface
    private int[] verifiers;            //Contains all the Inputs Validators of the Underlying Form Interface
    private int[] directions;
    private String[] buttons;           //Contains all the Buttons of the Underlying Form Interface
    private String[] buttonsLocation;
    private String[] message;           //Contains all the Prompt Messages of the Underlying Form Interface
    private String[] errorMessage;      //Contains all the Error Messages of the Underlying Form Interface
    private String[] messageLabel;

    private Hashtable<Integer,List<String>> fieldOptionsValue;
    private Hashtable<Integer,List<String>> fieldOptionsDisplay;
    /* The object(Entity) where the Form Interface store the data*/
    private Object formDataObject;
    /* The class of the underlying Entity which the Form Interface store the data*/
    private Class<?> formClass;
    private String formName;
    private Color formColor;
    /* No of the Columns in the underlying form*/
    private int gridSize;
    /* Whether the form is editable or not, Some forms are made Readonly while others are editable
     * This is set by the Business rules of the user*/
    private boolean isEdit=true;
    /* Whether mapping is used when the Labels ,Field Data are being read from the external file*/
```

```
53      private boolean mapping=false;
54
55      /*This is an In Memory Persistance Storage for the Entity(Collection Of Input
  Fields) that this class
56       *  represent through the Form Interface (GenericForm Class)*  */
57      private Repository repository;
58      private int newID;
59      /* Action Listener which will be used to Handle Action commands in the
  underlying form*/
60      private ActionListener actionListener;
61
62      public FormConfiguration(){}
63
64
65      // *************** All the Getters and Setters *************************//
66      public String[] getLabelNames() {
67          return labelNames;
68      }
69
70      public void setLabelNames(String[] labelNames){
71          this.labelNames=labelNames;
72      }
73
74      public String[] getFieldNames() {
75          return fieldNames;
76      }
77
78      public void setFieldNames(String[] fieldNames){
79          this.fieldNames=fieldNames;
80      }
81
82      public Class<?>[] getFieldClass() {
83          return fieldClass;
84      }
85
86      public void setFieldClass(Class<?>[] fieldClass){
87          this.fieldClass=fieldClass;
88      }
89
90      public int[] getFieldSize() {
91          return fieldSize;
92      }
93
94      public void setFieldSize(int[] fieldSize){
95          this.fieldSize=fieldSize;
96      }
97
98      public int[] getVerifiers() {
99          return verifiers;
100     }
101
102     public void setVerifiers(int[] verifiers) {
103         this.verifiers = verifiers;
104     }
105
```

```
106     public int[] getDirections() {
107         return directions;
108     }
109 
110     public void setDirections(int[] directions) {
111         this.directions = directions;
112     }
113 
114     public String[] getButtons() {
115         return buttons;
116     }
117 
118     public void setButtons(String[] buttons) {
119         this.buttons = buttons;
120     }
121 
122     public String[] getButtonsLocation() {
123         return buttonsLocation;
124     }
125 
126     public void setButtonsLocation(String[] buttonsLocation) {
127         this.buttonsLocation = buttonsLocation;
128     }
129 
130     public Class<?> getFormClass(){
131         return formClass;
132     }
133 
134     public void setFormClass(Class<?> formClass){
135         this.formClass=formClass;
136     }
137 
138     public Color getFormColor() {
139         return formColor;
140     }
141 
142     public void setFormColor(Color formColor) {
143         this.formColor = formColor;
144     }
145 
146     public String getFormName() {
147         return formName;
148     }
149 
150     public void setFormName(String formName) {
151         this.formName = formName;
152     }
153 
154     public String[] getMessageLabel() {
155         return messageLabel;
156     }
157 
158     public void setMessageLabel(String[] messageLabel) {
159         this.messageLabel = messageLabel;
160     }
```

```
161
162     public String[] getMessage() {
163         return message;
164     }
165
166     public void setMessage(String[] message) {
167         this.message = message;
168     }
169
170     public String[] getErrorMessage() {
171         return errorMessage;
172     }
173
174     public void setErrorMessage(String[] errorMessage) {
175         this.errorMessage = errorMessage;
176     }
177
178     public boolean isMapping() {
179         return mapping;
180     }
181
182     public void setMapping(boolean mapping) {
183         this.mapping = mapping;
184     }
185
186     public boolean isEdit() {
187         return isEdit;
188     }
189
190     public void setEdit(boolean isEdit) {
191         this.isEdit = isEdit;
192     }
193
194     public Object getFormData(){
195         return formDataObject;
196     }
197
198     public void setFormData(Object formDataObject){
199         this.formDataObject=formDataObject;
200     }
201
202     public Hashtable<Integer, List<String>> getFieldOptionsValue() {
203         return fieldOptionsValue;
204     }
205
206     public void setFieldOptionsValue(Hashtable<Integer,List<String>>
    fieldOptionsValue){
207         this.fieldOptionsValue=fieldOptionsValue;
208     }
209
210     public Hashtable<Integer, List<String>> getFieldOptionsDisplay() {
211         return fieldOptionsDisplay;
212     }
213
214     public void setFieldOptionsDisplay(Hashtable<Integer,List<String>>
```

```
fieldOptionsDisplay){
215         this.fieldOptionsDisplay=fieldOptionsDisplay;
216     }
217 
218     public int getGridSize(){
219         return gridSize;
220     }
221 
222     public void setGridSize(int gridSize){
223         this.gridSize=gridSize;
224     }
225 
226     public ActionListener getActionListener() {
227         return actionListener;
228     }
229 
230     public void setActionListener(ActionListener actionListener) {
231         this.actionListener = actionListener;
232     }
233 
234     public Repository getRepository(){
235         return repository;
236     }
237 
238     public void setRepository(Repository repository){
239         this.repository=repository;
240     }
241 
242     // *************** End Of All the Getters and Setters
**************************//
243 
244     /* Get the TotalInput Fields in the form */
245     public int getTotalFields(){
246         return labelNames.length;
247     }
248 
249     /* Get the size of the Message Labels in the Form */
250     public int getMessagesSize() {
251         if (messageLabel==null) return 0;
252         return messageLabel.length;
253     }
254 
255 
256     /* This  Method is called whenever GenericForm class need to add the Label
Component into the Form Layout
257      * Basically it create the JLabel component and return it to the Generic Form
Object*/
258     public JComponent getLabel(int index){
259         JLabel label=new JLabel(labelNames[index]);
260         label.setFont( new Font("Aerial", Font.PLAIN, 12));
261         return label;
262     }
263 
264     /* This is the Method basically creates the Input Field Component and set all
the necessary
```

```
265     * properties such as Field Size,Field Verifier etc. Also it populate all the
    options literals for the
266     * Radio Button,Combo Box etc */
267    public JComponent getFieldComponent(int index){
268        InputVerifier inputVerifier=null;
269        if (index !=0){
270            switch (verifiers[index]){
271            case 1 :    inputVerifier=new NumericVerifier();    //Selecting the
    right Pre existing Verifiers
272                        break;
273            case 2 :    inputVerifier=new TextVerifier();
274                        break;
275            }
276        }
277        /* Selecting the right Field class and create the appropriate Input Field
    and
278         * then setting the appropriate properties. */
279        if (fieldClass[index]==JTextField.class){
280            JTextField textField=new JTextField(fieldSize[index]);
281            textField.setInputVerifier(inputVerifier);
282            return textField;
283        }
284        else if (fieldClass[index]==RoundedTextField.class){
285            RoundedTextField roundedTextField=new
    RoundedTextField("",fieldSize[index]);
286            roundedTextField.setInputVerifier(inputVerifier);
287            return roundedTextField;
288        }
289        else if (fieldClass[index]==RoundedTextArea.class){
290            RoundedTextArea textArea=new RoundedTextArea(3,fieldSize[index]);
291            textArea.setLineWrap(true);
292            return textArea;
293        }
294        else if (fieldClass[index]==JTextArea.class){
295            JTextArea textArea=new JTextArea(3,fieldSize[index]);
296            textArea.setLineWrap(true);
297            return textArea;
298        }
299        else if (fieldClass[index]==RadioButtons.class){
300            int size=fieldOptionsDisplay.get(index).size();
301            String[] options=new String[size];
302            fieldOptionsDisplay.get(index).toArray(options);
303            return new RadioButtons(options,getDirections()[index],isEdit);
304        }
305        return null;
306    }
307
308    /* This  Method is called whenever GenericForm class need to add the Input
    Field Component
309     * into the Form Layout*/
310    public JComponent getInputField(int index){
311        JComponent field=getFieldComponent(index);
312        if (field.getClass()!=RadioButtons.class) {
313            if (fieldNames[index].equalsIgnoreCase("id")) {
314                field.setEnabled(false);
```

```
315             }
316             else {
317                 field=(JTextComponent )field;
318                 field.setEnabled(isEdit);
319             }
320         }
321         return field;
322     }
323 
324     /* This method is called Whenever Genericform class need to add the
    Messages(Prompt Message or Error Message)
325      * into the Form layout*/
326     public JComponent getMessageLabel(int index){
327         JLabel label=null;
328         if (messageLabel[index].equalsIgnoreCase("message")){
329             label=new JLabel(message[0]);
330             label.setFont( new Font("Aerial", Font.PLAIN, 12));
331             label.setForeground(Color.BLUE);
332         }
333         else if (messageLabel[index].equalsIgnoreCase("error"))
334         {
335             label=new JLabel(errorMessage[0]);
336             label.setFont( new Font("Aerial", Font.ITALIC, 12));
337             label.setForeground(Color.RED);
338             label.setVisible(false);
339         }
340         return label;
341     }
342 
343     /* This  Method is called whenever GenericForm class need to add the Buttons
    into the Form Layout*/
344     public JComponent getButton(int index, ActionListener listener){
345         JButton button=new JButton(buttons[index]);
346         button.setActionCommand(buttons[index+1]);
347         button.addActionListener(listener);
348         if (!(formName.equals("Logon" )|| formName.equals("Search") )){
349             button.setVisible(false);;
350         }
351         return button;
352     }
353 
354     /*This method return the value of the Component , If it is TextComponent , it
    return
355      * the contents of the TextComponent , If it is Radio Buttons, it return the
    the selected value.*/
356     public String getFieldValue(JComponent component, int index){
357 
358         if ((fieldClass[index]==JTextField.class) ||
    (fieldClass[index]==RoundedTextField.class) ||
    (fieldClass[index]==JTextArea.class) ||
    (fieldClass[index]==RoundedTextArea.class)){
359             return ((JTextComponent)component).getText();
360         }
361         else if (fieldClass[index]==RadioButtons.class){
362             String displayValue=((RadioButtons)component).getSelectedValue();
```

```
363             List<String> displayList=fieldOptionsDisplay.get(index);
364             int indexValue=displayList.indexOf(displayValue);
365             List<String> valueList=fieldOptionsValue.get(index);
366             if (indexValue!=-1) return valueList.get(indexValue);
367         }
368         return null;
369     }
370 
371     /* This method initialise all the input Fields on the Form */
372     private void formIntialise(GenericForm form ){
373         /* Loop through all the Input Fields of the form*/
374         for (int i=0;i< fieldNames.length;i++){
375             try {
376                 /* Get the Input Field Component By calling method
    "getFieldComponent" of the GenericForm*/
377                 JComponent component =form.getFieldComponent(i);
378                 /*Using the Reflection get the Method "setText" to set the
    contents of the input field */
379                 Method setTextMethod=component.getClass().getMethod("setText",
    String.class);
380                 if (component instanceof RadioButtons){
381                     ((RadioButtons) component).initialiseButtons();
382                 }
383                 else setTextMethod.invoke(component,"");        //invoke the
    setText method on the component
384 
385             } catch (Exception e) {
386                     e.printStackTrace();
387               }
388         }
389     }
390 
391     /*This Method bind the FormDataObject (Which is the Entity which store the
    data of the Form) with the
392      * Input Fields of the Form. Basically it synchronize the FormDataObject with
    the Input Fields of the Form */
393     public void bindForm(GenericForm form){
394         for (int i=0;i< fieldNames.length;i++){
395             /* If the FormDataObject is null ,It inititlise the Input Fields of
    the form */
396             if (formDataObject==null) {
397                 try {
398                     formIntialise(form );
399                     break;
400                 } catch (Exception e) {
401                     e.printStackTrace();
402                 }
403             }
404 
405             /* The following code block, Going through the properites of the
    Underlying FormDatObject(Entity)
406              * and read the value using the Getter of the Property and using the
    setText Method of Input field to
407              * populate its contents.
408              * As the Object type is unknown at compile time, It uses Reflection
```

```
   to access the getter and
409              * invoke it on the object appropriately  */
410             String methodName="get"+fieldNames[i];
411             try {
412
413                 Class<?>[] parameters={};
414                 Object[] paramData={};
415                 /* Access the Getter method of form data object(entity) by
   reflection */
416                 Method
   method=formDataObject.getClass().getDeclaredMethod(methodName, parameters);
417                 /* Invoke the  method on the form data object and save it as
   "Object"  Type*/
418                 Object obj =method.invoke(formDataObject, paramData);
419                 /* Get the respective Input field */
420                 JComponent component =form.getFieldComponent(i);
421                 /* Access the setText Method of Input field by Reflection */
422                 Method setTextMethod=component.getClass().getMethod("setText",
   String.class);
423                 if (component instanceof RadioButtons){
424                     obj=fromOptionValueToIndex(String.valueOf(obj),i);
425                 }
426                 /* Invoke the setText method of the Input field*/
427                 setTextMethod.invoke(component,String.valueOf(obj));
428
429             } catch (Exception e) {
430                     e.printStackTrace();
431               }
432         }
433     }
434
435     /* This method call the above method to bind the object with the form*/
436     public void bind(Object formDataObject,GenericForm form){
437         setFormData(formDataObject);
438         bindForm(form);
439     }
440
441     /* This method is called everytime form is saved using the save button*/
442     private void formSave(GenericForm form){
443             /* It loop through the input fields of the form and synchronizing the
   contents with the
444              * Form data object(Entity) by invoking the Setters dynamically*/
445             for (int i=0;i< fieldNames.length;i++){
446
447                 /* Get the input Field component and extract the  value of it*/
448                 JComponent component =form.getFieldComponent(i);
449                 String value=getFieldValue(component, i);
450                 /* Get the Setter method of the Property of the Object(Entity)*/
451                 String methodName="set"+fieldNames[i];
452
453                 Class<?>[] parameters=getMethodParameters(methodName);
454                 Object[] paramData=getMethodParametersValue(parameters, value);
455
456                 try {
457                     /* Get the Setter method dynamically and invoke on the
```

```
   Object(Entity) */
458                     Method
   method=formDataObject.getClass().getDeclaredMethod(methodName, parameters);
459                     method.invoke(formDataObject, paramData);
460
461                 } catch (Exception e) {
462                     e.printStackTrace();
463                   }
464             }
465             bindForm(form); //Call this method to synchronize the data with the
   form
466     }
467
468
469     /* This is the Event Handler for the FormSaveEvent. This Event is fired when
   buttons are clicked on the
470      * Form.  */
471     @Override
472     public void FormSaveOccured(FormSaveEvent event) {
473         GenericForm  form=(GenericForm)event.getSource();
474         /* If the underlying data object is null for the Form, Then create new
   Instance and
475          * Call the add method of the Repository class to add the newly created
   object(Entity) to
476          * the collection */
477         if (formDataObject==null) {
478             try {
479                 formDataObject=formClass.newInstance();
480                 if (repository!=null){
481                     repository.add(formDataObject);
482                 }
483             } catch (Exception e) {
484                 e.printStackTrace();
485             }
486         }
487         /* Save the input fields data of the form into the newly created object or
   existing object*/
488         formSave(form);
489     }
490
491     public int fromOptionValueToIndex(String value,int fieldIndex){
492         List<String> options=(List<String>)fieldOptionsValue.get(fieldIndex);
493         return options.indexOf(value);
494
495     }
496
497     /* The following two methods are used to get the parameters and their values
   for the
498      * Object's Method type using Reflection, These are being used in the method
   "formSave" */
499     public  Class<?>[] getMethodParameters(String methodName){
500
501         Method[] methods=formDataObject.getClass().getDeclaredMethods();
502         for (int i=0;i<methods.length;i++){
503             if (methods[i].getName().equals(methodName)){
```

```
504                 Class<?>[] parameters=methods[i].getParameterTypes();
505                 return parameters;
506             }
507         }
508         return null;
509     }
510     public Object[] getMethodParametersValue(Class<?>[] parameters,String value){
511         Object[] objectsValue=new Object[parameters.length];
512         int count=0;
513         for (int i=0;i<parameters.length;i++){
514             if (parameters[i]==Integer.TYPE){
515                 if ((value==null) || (value.equals("")) ){
516                     objectsValue[count]=0;
517                     if (newID!=0){
518                         objectsValue[count]=newID;
519                         newID=0;
520                     }
521                 }
522                 else objectsValue[count]=Integer.valueOf(value);
523             }
524             else objectsValue[count]=value;
525             count++;
526         }
527         return objectsValue;
528     }
529 }
530
```