```java
package project;

import java.awt.Color;

/* This will create FormConfiguration and GenericForm objects.
 * First it creates FormConfiguration objects for differnet forms by reading the form data
 * from the config.txt file and then creates the respective GenericForm object */
public class FormFactory {
    public static int unID=0;
    private String fileName;

    public FormFactory(String fileName){
        this.fileName=fileName;
    }

    /* Basically it takes a line of String and then parse into ConfigAttribute Object
     * MapToConfig method should create different formIds for differnt action.
     * For an example, if the action is mapping then it should create formID=100 or 200 or 300
     * depend on how many differnt mappings blocks the text file have. in our case , there are
     * 2 mapping blocks, so it should create attributes with formID=100,200
     * if The action is form Parsing , it will creates with formID=201,202,203 ........
     */
    public static  ConfigAttribute mapToConfig(String line) {
        String[] attributesArray=line.split("=");
        String[] attributeValues=attributesArray[1].split(",");
        String attributeName=attributesArray[0].trim();
        if (attributeName.equalsIgnoreCase("action")){
            if (attributeValues[0].trim().equalsIgnoreCase("formParsing")){
                FormFactory.unID++;
            }
            else
            {
                if (FormFactory.unID<100) FormFactory.unID=100;
                else if (FormFactory.unID<200) FormFactory.unID=200;
            }
        }

        ConfigAttribute configAttribute=new ConfigAttribute(attributeName.trim(), attributeValues,FormFactory.unID);
        return configAttribute;
    }

    /* This will creates Color object from  Integer array with 3 values for RGB */
    private Color mapToColor(String[] colorArray){

        return new Color(Integer.parseInt(colorArray[0]),Integer.parseInt(colorArray[1]),Integer.parseInt(colorArray[2]));
    }
```

```
64
65      /*This will map the String into class */
66      private static Class<?> mapToClass(String str){
67
68          switch (str){
69          case "1":
70              return JTextField.class;
71          case "2":
72              return JTextArea.class;
73          case "3":
74              return RoundedTextField.class;
75          case "4":
76              return RoundedTextArea.class;
77          case "5":
78              return RadioButtons.class;
79
80          }
81          return null;
82      }
83
84      /* This will creates List of String from  ConfigAttribute object by convering
    Array of
85       * String into List of String  */
86      private static List<String> maptoList (ConfigAttribute configAttribute){
87          List<String>
    attrtibuteValueList=Arrays.asList(configAttribute.getAttributeValue());
88          return attrtibuteValueList;
89
90      }
91
92      /* This will convert Array of String into Array of class */
93      private  Class<?>[] convertToclassArray(String[] array){
94
95          Class<?>[] classArray=new Class<?>[array.length];
96          List<Class<?>> classList= Arrays.asList(array).stream()
97                                      .map(FormFactory::mapToClass)
98                                      .collect(Collectors.toList());
99
100          classList.toArray(classArray);
101          return classArray;
102
103
104     }
105
106     /* This method creates collection of ConfigAttributes Objects by scanning an
    external file line by line
107      * and then parse into ConfigAttribute Objects using "mapToConfig" method. The
    external file is the storage
108      * where the data for the Forms stored*/
109     public   List<ConfigAttribute> csvParser(){
110         List<ConfigAttribute> configAttributes=null;
111         try {
112             InputStream is=new FileInputStream(new File(fileName));
113             BufferedReader br=new BufferedReader(new InputStreamReader(is));
114             configAttributes=br.lines()
```

```
115                     .filter(line -> !(line.trim().equals("")) &&  !
    (line.startsWith("**") ||line.endsWith("**")) )
116                     .map(FormFactory::mapToConfig)
117                     .collect(Collectors.toList());
118             br.close();
119         } catch (Exception e) {
120             e.printStackTrace();
121         }
122         return configAttributes;
123 
124     }
125 
126     /* This method create collection of FormConfiguration objects by using the
    "csvParser" Method
127      * There are several FormConfiguration objects will be created depend on the
    contents of the text file.
128      * If there are 5 form definitions defined in the file, there will be 5
    FormConfiguration
129      * objects will be created
130      * */
131     public List<FormConfiguration> createFormConfigurators (){
132 
133         List<FormConfiguration> configCollection=new
    ArrayList<FormConfiguration>();
134         List<ConfigAttribute> configAttributes=csvParser();
135         FormConfiguration config=new FormConfiguration();
136         Hashtable<Integer,List<String>> ht;
137         for (ConfigAttribute configAttribute :configAttributes ){
138             String attributeName=configAttribute.getAttributeName();
139             int formID=configAttribute.formID;
140             String[] attributesValues=configAttribute.attributeValue;
141             List<String> strList=Arrays.asList(attributesValues);
142             int[] intArray;
143             /* Following switch statments go through properties of
    FormConfiguration(config variable) and
144              * set the value by reading from the appropritae attribute of the
    ConfigAttribute Object */
145             switch (attributeName){
146                 case "labelNames":
147                     /* If the form data  use the setting "mapping=on" then
    attribute values (array of string)
148                      * of the Label Names will be mapped using the Mapping
    Service. Label mapping use the
149                      * formID=100 .*/
150                     attributesValues=MappingService.mapArray(configAttributes,attr
    ibutesValues,100, config.isMapping());
151                     config.setLabelNames( attributesValues);
152                     break;
153 
154                 case "fieldNames":
155                     /* If the form data  use the setting "mapping=on" then
    attribute values (array of string)
156                      * of the Field Names will be mapped using the Mapping
    Service. Field mapping use the
157                      * formID=200 */
```

```
158                     attributesValues=MappingService.mapArray(configAttributes,attr
    ibutesValues,200, config.isMapping());
159                     config.setFieldNames( attributesValues);
160                     break;
161                 /* The following code block setting properties of the
    FormConfiguration object*/
162                 case "fieldClass":
163                     config.setFieldClass(convertToclassArray( attributesValues));
164                     break;
165 
166                 case "buttons":
167                     config.setButtons( attributesValues);
168                     break;
169 
170                 case "location":
171                     config.setButtonsLocation( attributesValues);
172                     break;
173 
174                 case "messageLabel":
175                     config.setMessageLabel( attributesValues);
176                     break;
177 
178                 case "errorMessage":
179                     config.setErrorMessage( attributesValues);
180                     break;
181 
182                 case "message":
183                     config.setMessage( attributesValues);
184                     break;
185 
186                 case "gridSize":
187                     config.setGridSize(Integer.parseInt(attributesValues[0]));
188                     break;
189 
190                 case "formName":
191                     config.setFormName(attributesValues[0].trim());
192                     break;
193 
194                 case "mapping":
195                     if (attributesValues[0].trim().equalsIgnoreCase("on")){
196                         config.setMapping(true);
197                     }
198                     else config.setMapping(false);
199                     break;
200                 case "formClass":
201                         try {
202                             config.setFormClass(Class.forName(attributesValues[0])
    );
203                         } catch (ClassNotFoundException e) {
204                             e.printStackTrace();
205                         }
206                         break;
207 
208                 case "fieldSize":
209                     intArray= strList.stream()
```

```
210                                 .mapToInt(Integer::parseInt)
211                                 .toArray();
212
213                     config.setFieldSize(intArray);
214
215                 case "verifiers":
216                     intArray= strList.stream()
217                     .mapToInt(Integer::parseInt)
218                     .toArray();
219
220                     config.setVerifiers(intArray);
221                     break;
222
223                 case "directions":
224                     intArray= strList.stream()
225                     .mapToInt(Integer::parseInt)
226                     .toArray();
227
228                     config.setDirections(intArray);
229                     break;
230
231                 case "optionsV":
232                     intArray= strList.stream()
233                     .mapToInt(Integer::parseInt)
234                     .toArray();
235                      ht= getOptionsValues(intArray, configAttributes,formID);
236                     config.setFieldOptionsValue(ht);
237                     break;
238
239                 case "optionsD":
240                     intArray= strList.stream()
241                     .mapToInt(Integer::parseInt)
242                     .toArray();
243                      ht= getOptionsValues(intArray, configAttributes,formID );
244                     config.setFieldOptionsDisplay(ht);
245                     break;
246                 /* This is the last attribute of the Form data of the text file
247                  * This should complete the setting properties of FormCofiguration
    object and then add the
248                  * object to the collection, lastly it should start the same
    process by creating
249                  * new FormConfiguration object for the other Form Data in the
    text file and so on.*/
250                 case "color":
251                     config.setFormColor(mapToColor(attributesValues));
252                     configCollection.add(config);
253                     config=new FormConfiguration();
254             }
255
256         }
257         return configCollection;
258     }
259
260      /* This method read the option values  for different fields (Radio
    buttons,Combobox etc) from
```

```
261     * the text file. It read as a pair from the "intArray" . A pair (1,11) tell
    the method to read the contents
262     * of the "Options1" attribute(ConfigAttribute) as a List and assign to "Field
    11" */
263    public Hashtable<Integer,List<String>> getOptionsValues(int[]
    intArray,List<ConfigAttribute> configAttributes,int formID){
264        Hashtable<Integer,List<String>> ht=new Hashtable<Integer,List<String>>();
265        int count=0;
266        Predicate<ConfigAttribute> myFilter=null;
267        for (int i : intArray){
268            if (count % 2 ==0){
269                 myFilter= (att  -> att.getAttributeName().contains("options"+i));
270            }
271            if (count % 2 ==1){
272                List<String> list=configAttributes.stream()
273                            .filter(att -> att.formID==formID)
274                            .filter(myFilter)
275                            .map(FormFactory::maptoList)
276                            .flatMap(List::stream)
277                            .collect(Collectors.toList());
278                ht.put(i, (List<String>)list);
279            }
280            count++;
281        }
282        return ht;
283    }
284
285    /*This will Loop through the collection of FormConfiguration objects and
    create the appropriate
286     * Forms(Questionnaire,Serach,Logon,Participant etc)  using the
    FormConfiguration object(form Data)*/
287    public Map<String,GenericForm> createForms(){
288        unID=0;
289        Map<String,GenericForm> formCollection=new HashMap<String,GenericForm>();
290        formCollection.clear();
291        List<FormConfiguration> configuratorsList=createFormConfigurators();
292        for (FormConfiguration config : configuratorsList){
293            GenericForm form=new GenericForm(config);
294            formCollection.put(config.getFormName(), form);
295        }
296        return formCollection;
297    }
298
299    /* This will create the FormConfiguration of the form Name that is passed as a
    parameter
300     * For an example , if "Participant" is passed as a form Name it gives the
    FormConfiguration Object
301     * for that form */
302    public FormConfiguration createFormConfiguration(String formName){
303        unID=0;
304        List<FormConfiguration> configuratorsList=createFormConfigurators();
305        Optional<FormConfiguration> formConfigurator=configuratorsList.stream()
306                            .filter(configurator
    ->configurator.getFormName().equals(formName))
307                            .findFirst();
```

```
308         if (formConfigurator.isPresent()){
309             return formConfigurator.get();
310         }
311         else return null;
312     }
313 
314 }
315 
```